Языки логики и классы сложности. Рефлексия. $\mathbb{NP} \neq \mathbb{P}$.

Д. Л. Гуринович

28 сентябряы 2017 года

УДК 510.52

Содержание

Аннотация

Использованный метод

Данная заметка – «перевод» в термины «языков» и классов их сложности с терминов «задач» предыдущей заметки на эту тему – «Программа Гильберта, $\mathbb{NP} \neq \mathbb{P}$, Рефлексия» (https://edrid.ru/rid/217.015.93d1.html). «Переводя» прежнюю статью, я привожу её в соответствие с той терминологией, которая, как оказалось, принята при обсуждении вопросов сложности теории алгоритмов на математическом форуме.

Примеры процедур диагонализации, алгоритмов, формирующих и работающих со своим программным кодом (с примером работающей программы), подробное рассмотрение вопросов моделей интерпретации, доказательства для некоторых известных логических фактов (включая теорему о неопределимости, теорем Гёделя и т.п.) остались в прежней заметке, где их и можно почитать - в новую заметку я их не переносил.

Заодно были «переведены» кое-какие сведения из формальной логики в термины «языков» и классов сложности теории алгоритмов. А разные теоремы о неразрешимостях в таких терминах можно переписать как теоремы о несводимости языка одного класса сложности к другому классу сложности. В частности, теорема о неопределимости записывается в виде неравенства классов $\mathbb{NF} \neq \mathbb{F}$. А на базе этих «переведённых» результатов логики затем разбирается вопрос о принадлежности (сводимости) языка класса $\mathbb{NP}$ к классу $\mathbb{P}$ и получается доказательство для их неравенства: $\mathbb{NP} \neq \mathbb{P}$.

На уровне формальной логики идеи доказательства о несводимости классов сложности получаются более прозрачными – из-за отсутствия полиномиальных ограничений (требуется только финитность) и за счёт того, что «переписать» давно известные и проверенные результаты в терминах теории алгоритмов проще, чем открывать что-то заново. А результат о рефлексии интересен сам по себе и на уровне формальной логики он не отягощен дополнительными построениями.

Но если интересен только вопрс о $\mathbb{NP} \neq \mathbb{P}$, то можно начинать читать заметку сразу с пункта 6 раздела «Язык LS из класса $\mathbb{NP}$». До конца раздела (4 страницы) изложено построение несколько искусственного языка LS из класса $\mathbb{NP}$ и то, почему его невозможно свести к классу $\mathbb{P}$. То есть, фактически даётся в общих чертах доказательство неравенства $\mathbb{NP} \neq \mathbb{P}$.

А следующий раздел (чуть больше 6 страниц) – «$\mathbb{NP} \neq \mathbb{P}$ на примере языка LS» – лишь детализирует некоторые технические подробности в доказательстве неравенства $\mathbb{NP} \neq \mathbb{P}$. Доказательство весьма простое, как мне представляется – но желательно иметь представление о теории доказательств из математической логики и начальные представления о классах сложности $\mathbb{P}$ и $\mathbb{NP}$ из теории алгоритмов.

Четыре препятствия, которые мне было трудно преодолеть и из-за которых так долго выполнялась техническая, по сути, операция переписывания статьи с терминов задач в термины языков и классов их сложности:

1. Недоказуемость утверждения $AntiSol$ при не остановке $Sol(\overline{AntiSol})$; 2. Программный код алгоритма-решения – часть сертификата, а не слова для алгоритма проверки; 3. Правильное поведение $Sol(\overline{AntiSol})$ в заведомо не корректном случае (апелляция к недетерминированному алгоритму-решению); 4. Мои слабые способности в освоении новых для меня языков и формализмов.

1 «Стандартный» язык логики L. Прикладные роли языка.

Для начала добавим в формализм языков вот что: «прикладные роли языка», или сокращенно «роль языка». То, что постоянно используется, но не имело никакого названия (я не встречал, по крайней мере):

Язык – это просто множество слов, которое само по себе никакого отношения к алгоритмам не имеет. Другое дело, что можно задавать «внешние» отношения между языком и алгоритмами.

Например, алгоритм проверки, с помощью которого можно описать язык класса $\mathbb{NP}$ и отнести подобный язык к классу $\mathbb{NP}$ – вовсе не является «частью» языка. Язык можно описать и неполиномиально долгим по своей работе алгоритмом проверки и ещё бесконечным количеством способов. Язык от этого никак не изменится. Поэтому при отнесении языка к классу $\mathbb{NP}$ говорят, что соответствующий алгоритм для описания языка «существует», но такой способ описания данного языка вовсе не является обязательным или единственным.

Кстати, те же «сертификаты» («свидетели») тоже не являются частью языка из класса $\mathbb{NP}$ – это как раз атрибут некоторого алгоритма проверки, с помощью которого можно описать данный язык. Если бы сертификаты были частью языка, то он заведомо не мог бы относиться к классу $\mathbb{P}$ (вопрос о равенстве $\mathbb{NP}$ и $\mathbb{P}$ тогда не возникал бы) – так как среди алгоритмов, которые могут использоваться для описания языков класса $\mathbb{P}$, обязательно есть («существуют») и такие,

которым сертификаты не требуются.

Рассмотрим язык логики L – для теории Пеано, например. Этому языку соответствует алгоритм проверки

$\mathrm{Lc}(t, y)$, где t – текст теоремы, а y – текст доказательства. Результат работы алгоритма $\mathrm{Lc}(t, y)$ будет 1, если в t – текст утверждения и в y – текст его доказательства. И результат работы алгоритма $\mathrm{Lc}(t, y)$ будет 0 в ином случае.

Буква «с» в названии алгоритма – от слова «классический» - так как этот вариант языка логики и алгоритма проверки разрабатывался ещё в период формирования и активной реализации программы Гильберта. Но мы будем считать, что аргументы в алгоритм $\mathrm{Lc}(t, y)$ передаются не как гёделевы номера (числу миллион тогда соответствовал бы аргумент в миллион «счётных палочек»), а как тексты соответствующих утверждений и доказательств с «буквами» и «цифрами» - для позиционного представления чисел и формул.

С учётом последнего замечания (об аргументах) алгоритм $\mathrm{Lc}(t, y)$ работает полиномиальное время (количество шагов) относительно размера своих аргументов (от количества символов в них). Что не делает язык L языком класса $\mathbb{NP}$, разумеется – так как нет (это доказано для общего случая из теорем Гёделя) полиномиальной (и вообще никакой) связи между размерами теорем и размерами их доказательств.

И данный алгоритм $\mathrm{Lc}(t, y)$ может рассматриваться в качестве полной прикладной роли языка L.

Прикладную роль для данного языка назовём полной, если для каждого слова t данного языка есть соответствующая ему единичная задача с положительным решением $\mathrm{Lc}(t, y) = 1$ для некоторого «доказательства» y теоремы t. И нет никакой единичной задачи с положительным решением для слова t, которое не принадлежит языку. То есть - взаимно-однозначное соответствие между некоторым множеством решаемых задач и множеством слов языка. Для некоторых языков (из класса $\mathbb{P}$, например) в качестве «доказательства» y может выступать и само слово t.

У нас есть множество задач, задаваемых алгоритмом $\mathrm{Lc}(t, y)$ для утверждений t, которые оказываются теоремами, если есть соответствующее доказательство y – при котором имеем $\mathrm{Lc}(t, y) = 1$. Имеется взаимно-однозначное соответствие между языком L и задачами, задаваемыми утверждениями t, для которых имеется y такой, что $\mathrm{Lc}(t, y) = 1$. И это – 1-я (Первая) прикладная роль разбираемого нами языка L.

2 Алгоритмы-решения. Теорема о неопределимости. Утверждение *AntiSol*.

Но в то же время язык L содержит множество теорем о свойствах чисел и алгоритмов. Ведь теория Пеано в состоянии «представлять» алгоритмы и позволяет формулировать и доказывать теоремы о них. В частности, имеется теорема о неопределимости. Которая даёт ответ (негативный) на вопрос, может ли какой-либо алгоритм «представить» всё множество теорем теории Пеано. Или, иными словами, представить язык L.

Речь идет о таком алгоритме Sol(t), который должен был бы возвращать 1 в том случае, если аргумент t является словом языка L и 0 – в противном случае. Слово будет принадлежать языку L в том и только том случае, если t – текст теоремы теории Пеано. Предполагается, что есть способ сопоставить результат (или его отсутствие) работы любого алгоритма Sol(t) для любого слова t с фактом (не)принадлежности данного слова языку L. В этом сопоставлении язык L играет роль «эталона» с которым сравниваются результаты (или отсутствие результата) произвольного алгоритма Sol(t). Эту роль языка L назовём 2-й (Второй) прикладной ролью языка L.

Разбираемые алгоритмы (Sol(t) в данном примере) из 2-й прикладной роли языка будем называть «алгоритмы-решения», что не означает, что они все или какой-то из них способен представить язык L – соответствовать этому «эталону». Но если какой-то алгоритм удовлетворяет требованиям 2-й роли – (всегда останавливается с результатом и результат всегда соответствует «эталону») назовём его «корректный алгоритм-решение».

Перечислим свойства, которыми должен обладать корректный алгоритм-решение, но перечислим их в виде отрицания данных свойств. Потому что произвольный алгоритм-решение Sol(...) не обладает этими свойствами на утверждении *AntiSol* (которое строится в зависимости от выбранного алгоритма-решения). И приводимое ниже отрицание свойств корректности для алгоритма-решения Sol(...) будет, вместе с тем, свойствами утверждения *AntiSol*:

1. Не верно, что если *AntiSol* является теоремой теории Пеано (то есть, словом в языке L), то алгоритм $Sol(\overline{AntiSol})$ возвращает за конечное время подтверждение этого факта, то есть $Sol(\overline{AntiSol}) = 1$.

Сразу оговоримся, что записи вида $\overline{AntiSol}$ или $\overline{\text{Sol}(...)}$ обозначают текст утверждения *AntiSol* на заранее оговоренном языке логики и программный код (тоже текст, фактически) алгоритма Sol(...), записанный оговорённым способом представления алгоритмов в выбранной за основу языка LS теории первого порядка (теория Пеано способна представлять алгоритмы, но не очень наглядно – тут есть что улучшать для практических нужд).

2. Не верно, что если *AntiSol* НЕ является теоремой теории Пеано (то есть, не является словом в языке L), то алгоритм $Sol(\overline{AntiSol})$ возвращает за конечное время результат о непринадлежности слова *AntiSol* языку L: $Sol(\overline{AntiSol}) = 0$.

Я напомню простую идею доказательства теоремы о неопределимости и то, что это за утверждение - *AntiSol*:

Строится алгоритм ContrSol(), который генерирует в текстовом виде утверждение со своим собственным программным кодом (с помощью процедуры диагонализации) $\overline{\text{ContrSol}() = 1}$ и запускает алгоритм Sol($\overline{\text{ContrSol}() = 1}$). Если запущенный алгоритм Sol($\overline{\text{ContrSol}() = 1}$) вернёт любой результат, кроме 0, то алгоритм ContrSol() вернёт 0. А если запущенный алгоритм Sol($\overline{\text{ContrSol}() = 1}$) вернёт 0, то алгоритм ContrSol() вернёт 1. А если запущенный алгоритм

$\mathrm{Sol}(\overline{\mathrm{ContrSol}() = 1})$ не остановится, то алгоритм ContrSol() тоже не остановится.

Утверждение *AntiSol* – это вот что: ContrSol() = 1.

Разумеется, алгоритм $Sol(\overline{AntiSol})$ не может подтвердить утверждение *AntiSol* результатом 1– так как тогда будет верным ContrSol() = 0, что является отрицанием «подтверждённого» результата. Но алгоритм $Sol(\overline{AntiSol})$ не может и отвергнуть утверждение *AntiSol* результатом 0 – так как тогда будет верным ContrSol() = 1, что является доказательством *AntiSol* (не напрямую, но доказательство можно построить).

Единственное, чего нам не хватает в свойствах утверждения *AntiSol* – того, чтобы в случае, если алгоритм $Sol(\overline{AntiSol})$ не останавливался, то утверждение *AntiSol* не было словом языка L (не доказывалось в нём).

Для чего нам нужно это свойство утверждения *AntiSol*? Для того, чтобы всё указывало алгоритму $Sol(\overline{AntiSol})$ на отсутствие для него возможности найти подтверждение (доказательство) утверждения *AntiSol*. Понятно, что сам он не может найти доказательство утверждения *AntiSol* в непротиворечивой теории и вернуть информацию (результат 1) об этом– иначе сразу было бы противоречие. Но в некоторых даже непротиворечивых теориях для некоторых алгоритмов $Sol(\overline{AntiSol})$, которые не останавливаются, может быть доказано соответствующее им утверждение *AntiSol*. И тогда отсутствие остановки (и результата) у алгоритма $Sol(\overline{AntiSol})$ не было бы однозначным расхождением с логикой теории – хоть доказательство сам алгоритм $Sol(\overline{AntiSol})$ найти и не смог, но и 0 он не вернул, потому что утверждение *AntiSol* всё же имеет доказательство!

Однако такую возможную неоднозначность легко устранить небольшой модернизацией алгоритма ContrSol():

Помимо запуска алгоритма $Sol(\overline{AntiSol})$ алгоритм ContrSol() запускает параллельно ещё и перебор всех доказательств теории. И поступит наоборот, если будет в процессе перебора обнаружено доказательство *AntiSol* (то есть – вернёт 0). Тогда в непротиворечивой теории утверждение *AntiSol* не будет доказано, если $Sol(\overline{AntiSol})$ не остановится.

Поэтому теперь формулируем третье свойство для утверждения *AntiSol*:

3. Верно, что если алгоритм $Sol(\overline{AntiSol})$ не останавливается, то утверждение *AntiSol* НЕ является теоремой теории Пеано (то есть, не является словом в языке L).

Аналогично, при решении вопроса $\mathbb{NP}$ vs $\mathbb{P}$ необходимо решить вопрос о конкретном алгоритмерешении $\mathrm{Sol}_{\mathrm{NP}}(t)$ с полиномиальным относительно размера слова языка L_{NP}, временем работы. И вопрос этот в том, соответствуют ли результаты работы $\mathrm{Sol}_{\mathrm{NP}}(t)$ языку L_{NP}. И это соответствие для сравнения языка и результатов работы алгоритма – тоже 2-я (Вторая) прикладная роль разбираемого языка.

Притом эта 2-я прикладная роль «персонализирована» в отношении каждого проверяемого алгоритма, так как проверку на соответствие потенциально можно провести в отношении любого конкретного алгоритма. И, значит, 2-я прикладная роль подразумевает возможность однозначной идентификации конкретного алгоритма – вплоть до получения его программного кода. И такая возможность является принципиальной для нашего дальнейшего исследования.

Первая прикладная роль для языка L_{NP} та, которая определяет его принадлежность к классу сложности $\mathbb{NP}$. Некий алгоритм $\mathrm{Lc}_{\mathrm{NP}}(t, y)$. Подобные алгоритмы будем называть далее «алгоритм проверки».

Если для языка L_{NP} нет ни одного подходящего $\mathrm{Sol}_{\mathrm{NP}}(t)$, то язык L_{NP} не является языком

из класса $\mathbb{P}$ и неравенство $\mathbb{NP} \neq \mathbb{P}$ доказано.

Но если бы имелся алгоритм $\mathrm{Sol}_{\mathrm{NP}}(t)$, полиномиально быстрый относительно размера $|t|$ и результаты которого соответствуют языку L_{NP}, то данный алгоритм позволил бы задать 3-ю (Третью) прикладную роль для языка L_{NP}. И эта прикладная роль была бы полной и доказала (по определению) принадлежность языка L_{NP} классу сложности $\mathbb{P}$.

По аналогии с теоремой о неопределимости для языка L из логики, можно ожидать аналогичной теоремы и для языка L_{NP}, а вместе с этим и доказательства неравенства $\mathbb{NP} \neq \mathbb{P}$. Но прежде чем пытаться переносить свойства языка L на некий язык L_{NP} – попробуем «перевести» формальную логику и язык L к тому виду, который используется в теории алгоритмов.

Заметим, что информация (теорема) о невозможности для алгоритма доказать некоторое утверждение имеется в логике и это – следовательно – представлено непротиворечивым образом и может быть выражено в неком языке.

3 Язык логики LA

В каком же языке логики (не классическом) LA это может быть выражено? В том же самом языке – если смотреть на «слова» (теоремы) которые принадлежат и утверждения, которые не принадлежат (не доказаны), как и в зыке L. Будем использовать разные обозначения для этих языков только для того, чтобы обозначить, что у языка LA другая 1-я прикладная роль и другой алгоритм проверки, чем мы рассматривали для языка L.

Не трогая язык, мы изменим алгоритм проверки (с $\mathrm{Lc}(t,y)$ на $\mathrm{La}(t,a,y)$). Добавим сертификатов так, чтобы язык не изменился. Добавим, возможно, слов, которые не принадлежат языку, но с которыми работает алгоритм проверки. Затем для единичных задач рассмотрим «персональные» наборы сертификатов, в которых выбросим некоторые из тех подтверждающих сертификатов, которые заведомо не используются в силу теоремы о неопределимости.

Прежние сертификаты мы оставим без изменения, но записывать их будем в двух аргументах для алгоритма проверки: a, y. Где новая часть сертификата в аргументе a – это аргумент для программного кода алгоритма-решения. И для старых сертификатов этот аргумент пуст, но обозначать это будет так: $\overline{-}$. Можно было бы и пустую строку использовать: «», но мы выберем менее «слепой» вариант. В аргументе y по-прежнему находится текст, который в корректном случае является текстом доказательства.

Новые сертификаты тоже будут разбиты на 2 аргумента: a, y. Но в аргументе a будет программный код алгоритма-решения, который выдаёт ответ о (не)принадлежности языку LA проверяемого слова (утверждения) t.

Наш новый алгоритм проверки - алгоритм $\mathrm{La}(t,a,y)$. Он же – 1-я прикладная роль (полная!) для языка LA. Работает он так: «Очень быстро» возвращает 0 для случаев вида: $\mathrm{La}(\overline{AntiSol}, \overline{\mathrm{Sol}(...)}, y)$, а в остальном работает как $\mathrm{Lc}(t,y)$.

Что означают слова «очень быстро» в предыдущем абзаце? Они означают – за полиномиальное количество шагов относительно размеров $|t|$, $|a|$. При этом – независимо от доказательства в аргументе y и его размера.

Теперь, допустим, мы рассматриваем (произвольный) алгоритм-решение $\mathrm{Sol}(t)$. Какие слова и сертификаты должны быть сопоставлены данному алгоритму из первой прикладной роли языка LA? А вот такие:

t, $\overline{\mathrm{Sol}(...)}$

А всё дело в том, что само использование алгоритма $\mathrm{Sol}(t)$ для решения создаёт ту данность, которая в нашем языке LA выражена частью сертификата $a = \overline{\mathrm{Sol}(...)}$:

Для всех утверждений t, кроме $\overline{AntiSol}$, нет никакой разницы между сертификатом обычным $(\overline{-}, t)$ и «персонализированным» $(\overline{\mathrm{Sol}(...)}, t)$ – алгоритм проверки работает с ними одинаково.

1. Теперь разберём случай $Sol(\overline{AntiSol})$ и покажем, что никакие доказательства для $AntiSol$ ему недоступны ни в 1-ой прикладной роли языка L, ни в 1-й прикладной роли языка LA – при соответствии с алгоритмом-решением только тех сертификатов, часть a в которых равна программному коду алгоритма-решения.

1.1. Алгоритм-решение может быть корректным и может быть не корректным. Если он не останавливается, то он не корректен. Если он останавливается с результатом, отличным от 1 и 0, то он не корректен. Если он останавливается с 1 (единицей), когда проверяет доказуемость

утверждения t, но нет ни одного доказательства утверждения t, то он не корректен. И если он останавливается с 0 (нулём), а есть доказательство для утверждения t, то он не корректен. И если есть хоть одно утверждение t, на котором он не корректен, то он не корректен. В ином случае он корректен.

Поэтому на утверждении $AntiSol$ алгоритм Sol(...) не корректен.

1.2. Некорректность алгоритма-решения Sol(...) соответствует тому, что он не может найти правильное решение утверждению $AntiSol$. То есть, ему соответствуют лишь те «сертификаты» (доказательства), которые не подтверждают (не доказывают) утверждение $AntiSol$.

И с точки зрения недетерминированной машины-решения Тьюринга алгоритм $Sol(\overline{AntiSol})$ соответствует тем «разветвлениям» этой недетерминированной машины-решения, которые ведут вычисления для неподтверждающих «сертификатов» (доказательств) и правильный ответ для этих разветвлений – 0 (Ноль) – «не могу найти доказательство».

1.3. И в 1-й прикладной роли языка LA мы собрали все эти сертификаты для алгоритма $Sol(\overline{AntiSol})$ в совокупность всех сертификатов, соответствующих правильной работе $Sol(\overline{AntiSol})$. И часть a у этих сертификатов равна $\overline{\text{Sol}(...)}$ и они все не подтверждают утверждение $AntiSol$:

И не подтверждают они потому, что одни сертификаты (которые могли бы подтвердить утверждение $AntiSol$) не доступны $Sol(\overline{AntiSol})$ и в силу этого в данном контексте не подтверждают утверждение $AntiSol$, а доступные для $Sol(\overline{AntiSol})$ сертификаты так и остаются не подтверждающими. Поэтому построенная нами 1-я прикладная роль языка LA полностью соответствует исходной прикладной роли и тому языку, которому соответствовала исходная прикладная роль языка L. Язык не меняется ($LA = L$), построенный алгоритм проверки ему соответствует. И каждому алгоритму-решению можно ставить в соответствие лишь те сертификаты в новой прикладной роли, у которых часть a равна программному коду данного алгоритма-решения.

1.4. Если ставится задача, чтобы $Sol(\overline{AntiSol})$ нашёл ответ, который у него есть за конечное количество шагов от алгоритма проверки языка LA, то алгоритм проверки даёт его для $Sol(\overline{AntiSol})$ и этот ответ – ноль: $\text{La}(\overline{AntiSol}, \overline{\text{Sol}(...)}, ...) = 0$. При этом на месте аргумента доказательства стоит многоточие – алгоритм проверки не обращается в своей работе к данному третьему аргументу при таких 1-м и 2-м аргументах и в данном случае может быть записан в виде алгоритма с двумя аргументами $\text{La}(\overline{AntiSol}, \overline{\text{Sol}(...)}) = 0$.

1.5. Модифицированный алгоритм проверки в LA - это ведь лишь «ускорение» обычного алгоритма проверки языка L – не персонализированного. И то, что алгоритм-решение не мог найти в L – он не может найти и в LA, но теперь нет нужды бесплодно перебирать доказательства (сертификаты) в поисках доказательства утверждения $AntiSol$. Теперь все эти сертификаты, доступные для Sol(...), сразу НЕ подтверждают утверждение $AntiSol$, в соответствии с теоремой о неопределимости.

Далее упомяну о некоторых возникающих «сопутствующих» вопросах.

2.1. О возможных результатах работы алгоритма $Sol(\overline{AntiSol})$.

«Он не сможет найти» - правильная информация. Но когда «он» останавливается с данным результатом («не могу найти» - 0), то доказательство находится. Это и означает, что «он» не полон. То есть – любой алгоритм-решение не полон, так как «он» произволен.

А если заведомо не способный $Sol(\overline{AntiSol})$ не останавливается, чтобы сохранить неопределённость в том, принадлежит или не принадлежит утверждение $AntiSol$ языку LA? В LA

это ему заведомо не удастся – с учетом пункта 3 раздела «Алгоритмы-решения. Теорема о неопределимости. Утверждение $AntiSol$»:

Вечное затягивание с ответом ничего не даст - $Sol(\overline{AntiSol})$ в итоге всё равно не смог дать ответ «не могу найти доказательство», которого действительно нет – что является доступной «персонализированной» для алгоритма Sol(...) информацией в алгоритме проверки: $\mathrm{La}(\overline{AntiSol}, \overline{\mathrm{Sol}(...)}, ...) = 0$

2.2. Зависимость подтверждения слова сертификатом от программного кода алгоритма-решения – не приводит ли к «порче» алгоритмов-решений?

А разве не что угодно можно поставить в зависимость от алгоритма-решения в алгоритме проверки, чтобы алгоритм проверки возвращал «Не могу найти доказательство»? Тут есть два нюанса и первый в том, что от сертификата принадлежность слова не зависит. Просто если его подтверждает не этот, то - другой сертификат. И так же решается вопрос о том, как же принадлежность слова языку может зависеть от алгоритма. Не зависит. Это алгоритм может быть не способен подтвердить слово.

А второй нюанс в том, что реализация сертификатов и 1-ой прикладной роли будет не корректной, если алгоритм-решение якобы не может использовать верный сертификат, а он по факту выдаёт верный подтверждающий результат. Эта некорректность аналогично ошибочной 1-й прикладной роли – когда для имеющегося в языке слова нет подтверждающего сертификата для ошибочного алгоритма проверки.

Но мы же предполагаем, что наш алгоритм проверки построен корректно – то есть, его сертификаты действительно правильно указывают на слово и в части кода алгоритмов-решений тоже не содержат ошибок. И да, мы строили 1-ю прикладную роль языка LA на основе корректной прикладной роли языка L и теоремы о неопределимости. Поэтому прикладная роль языка LA – корректна.

Алгоритм – он интересен только как производитель результата. И поэтому его программный код идет только в дополнение к другой части сертификата - доказательству, а доказательство представлено и в независимом от программного кода алгоритма сертификате – при $a = \overline{-}$.

2.3. Почему алгоритм $Sol(\overline{AntiSol})$ не может вернуть правильный результат про утверждение $AntiSol$, если разбираться в причинах этого не формально?

А откуда возникает обсуждаемая неспособность алгоритма Sol(...)? Алгоритм Sol(...) не констатирует, а делает слово $\overline{AntiSol}$ принадлежащим или не принадлежащим языку LA. И если считать принадлежностью слова $\overline{AntiSol}$ языку то, что происходит после этого действия алгоритма Sol(...), то для Sol(...) слово $\overline{AntiSol}$ действительно ещё не принадлежит.

Алгоритм проверки выдаёт не один из 2-ух вариантов «есть доказательство для данного утверждения» и «нет доказательства для данного решения». Нет, смысл 0 (нуля) в ответе алгоритма проверки – «данный текст не является решением (доказательством) разбираемого утверждения», а вовсе не «нет решения у разбираемого утверждения» и между этими двумя смыслами есть и такой: «Для алгоритма Sol(...) нет решения у разбираемого утверждения».

Чем отличается «это доказательство не годится» от «этот алгоритм не годится»? И как алгоритм-решение может оказаться в состоянии, которое отличается и от «есть решение данного утверждения» и «нет решения данного утверждения»?

А дело в том, что некоторые алгоритмы расположены на причинно-следственной оси ДО ответа о принадлежности данного слова к языку. А в теории, кстати, бывает, что не достигнуто

ни одно из этих состояний – с логической точки зрения для данной теории – и это известно ещё из 1-ой теоремы Гёделя. Но фактически, конечно, факт отсутствия доказательства будет иметь место.

Просто «фактическая» сторона – это уже информация об арифметике, которая – как известно, полна, но это уже не теория 1-го порядка. И она даже не является эффективно аксиоматизируемой. Но даже с фактической точки зрения «решения не будет» лишь «после» того, как алгоритм Sol(...) не остановится, хоть это и бесконечный процесс. Процесс бесконечный, но отсутствие доказательства будет следствием того, что процесс бесконечный, то есть с причинно-следственной точки зрения факт отсутствия доказательства расположен «после» бесконечной работы алгоритма Sol(...).

2.4. Почему раньше в учебниках по теории алгоритмов не рассматривалось соответствие между программным кодом алгоритма-решения и соответствующими ему сертификатами алгоритма проверки?

Исследователям было очевидно, что аргумент, переданный алгоритму-решению, должен соответствовать слову, которое (не)принадлежит языку. Но упускали из вида, что сам алгоритм-решение и его программный код – ничуть не в меньшей степени данность, чем переданный аргумент и эта данность тоже должна как-то учитываться, как влияющая на результат поиска принадлежности слова к языку.

То есть, в достаточно выразительном языке программный код алгоритма-решения как-то влияет на то, какие слова принадлежат языку или влияет хотя бы на то, какой у него доступ к сертификатам, которые используются в 1-ой прикладной роли языка, на которую должен опираться данный алгоритм-решение.

И, кстати, зависимость принадлежности слов к языку от алгоритма-решения действительно есть – что было отмечено в данном разделе в первом абзаце после абзаца, пронумерованного номером 2.3.

Фигурально выражаясь – «быть Джоном Малковичем» - это не только получать почту Джона Малковича, пиццу Джона Малковича и зарплату на карточку Джона Малковича – то есть получать некие «значения» в «аргументы» Джона Малковича, а не в «аргументы» Брюса Уиллиса. И «быть Джоном Малковичем» – это не только (в дополнение к сказанному) оставлять на киноплёнке изображения Джона Малковича (а не Брюса Уиллиса) – результат работы Джона Малковича.

Но, помимо всего сказанного, «быть Джоном Малковичем» - это ещё иметь лицо, мозги и прочие системы организма Джона Малковича (а не Брюса Уиллиса), потому что именно это лицо, мозги и прочие его системы организма определяют те изображения на киноплёнке Джона Малковича (а не Брюса Уиллиса), которые и являются результатом работы Джона Малковича.

4 Рефлексия на языке LA. $\mathbb{NF} \neq \mathbb{F}$

Язык логики и его свойства очень важны для математики, а язык LA является одной из разновидностей языка логики. Поэтому выразим одно его важное свойство, позаимствовав понятие «класс сложности» из теории алгоритмов:

Язык LA принадлежит классу сложности $\mathbb{NF}$. Это аналог класса $\mathbb{NP}$, только без полиномиальных ограничений. Главное, чтобы алгоритм проверки работал конечное время (был финитным). Наш алгоритм проверки $\mathrm{La}(t, a, y)$ можно построить так, чтобы он работал полиномиальное время относительно размеров $|t|$, $|a|$, $|y|$.

Но в логике время не является принципиальным, а в класс $\mathbb{NP}$ язык LA при полиномиально быстром алгоритме $\mathrm{La}(t, a, y)$ не попадет, потому что полиномиального ограничения размеров сертификата $|a| + |y|$ от размеров слова $|t|$ нет в общем случае, так как размер корректного доказательства может быть каким угодно – независимо от размера доказываемого утверждения.

В логике интересен только сам факт остановки или не-остановки алгоритма – можно хоть гёделевыми номерами пользоваться вместо текстов, превращая ситуацию в явно не полиномиальную. А в теории алгоритмов гёделевы номера неприемлемы, равно как непригодна и стандартная модель интерпретации арифметики – для теории алгоритмов нам нужны тексты и разрядное представление для чисел, а модель интерпретации нужна «бытовая» (компьютерная) – с обычной записью тех же чисел, а не в виде «счётных палочек». Но это «лирическое отступление».

Так вот, язык LA принадлежит классу сложности $\mathbb{NF}$, а если бы он был разрешим, то его можно было отнести и к классу сложности $\mathbb{F}$ – финитных языков (к ним относится, например, логика высказываний), когда о слове за конечное время можно сказать, принадлежит оно языку или нет. Класс $\mathbb{F}$ – аналог класса $\mathbb{P}$, но без полиномиальных ограничений. Мы доказали, что LA не сводится к классу $\mathbb{F}$, так как для любого алгоритма-решения $\mathrm{Sol}(t)$ есть аргумент:

$\overline{AntiSol}$, о которых алгоритм

$Sol(\overline{AntiSol})$ не может получить корректного ответа как о слове $\overline{AntiSol}$ (не)принадлежащем языку LA. То есть, доказано:

$\mathbb{NF} \neq \mathbb{F}$

И это неравенство – обобщенный аналог теоремы о неопределимости.

Это – та же теорема о неопределимости из логики, но переписанная в терминах языков и классов сложности. И неразрешимость тут представлена как невозможность языку из одного класса сложности ($\mathbb{NF}$) принадлежать другому классу сложности ($\mathbb{F}$) – несводимость одного класса сложности к другому. Да и множество других теорем о неразрешимости могут быть переписаны как несводимость классов сложности друг к другу.

Но в процессе анализа мы получили более интересную – с мировоззренческой точки зрения – информацию: От корректной системы требуется понимание – кто она такая. И без этого понимания невозможно решение некоторых важных задач. То есть, мы вышли на формализацию такого важного для многих живых систем свойства как рефлексия.

Это как раз то, похоже, чего не хватало математике для анализа «разумных систем» - типа человека, коллективов и т.п. Ведь в соответствии с тезисом Чёрча их можно рассматривать как алгоритмы, а про алгоритмы математические теории в состоянии давать доказуемые ограничивающие утверждения. И вместе с рефлексией это оказывается принципиальной возможностью:

Теория (достаточно выразительная) не может доказать ограничивающее утверждения о себе в целом – это известно из 2-й теоремы Гёделя (теория не отличает себя от противоречивой теории, в которой доказано всё). Но внутри теории вполне доказаны ограничивающие теоремы об алгоритмах. Та же теорема о неопределимости. И эти сведения алгоритмам – если они достаточно выразительны – доступны.

Трудно переоценить важность рефлексии для жизни: Без понимания того, какие опасности грозят именно тебе и какие твои действия могут привести к катастрофе именно тебя – твоя жизнь не была бы сколько-нибудь продолжительной. И возможность формализации данного свойства (рефлексии) в математике – более принципиальна, на мой взгляд, чем неравенство $\mathbb{NF} \neq \mathbb{F}$ или $\mathbb{NP} \neq \mathbb{P}$.

Ещё один интересный момент – когда для конкретного алгоритма-решения в применённом методе использования 1-й прикладной роли языка LA задаётся тот набор сертификатов, который соответствует именно этому алгоритму-решению, а другие сертификаты отбрасываются. Дело в тома, что «быть кем-то» (в данном случае – быть данным конкретным алгоритмом-решением) – это не означает, что ты имеешь свойства, которые не имеет остальной мир: Мир содержит тебя вместе со всеми твоими свойствами. Значит, быть кем-то – это не иметь чего-то такого, что есть в мире, но при этом отделено от тебя какой-то границей.

5 Язык LS из класса $\mathbb{NP}$.

5.1 Язык LS, пункты 1-5

Теперь осталось на базе языка LA из класса сложности $\mathbb{NF}$ построить язык LS из класса $\mathbb{NP}$ и доказать на его примере неравенство $\mathbb{NP} \neq \mathbb{P}$ по аналогии с доказательством неравенства $\mathbb{NF} \neq \mathbb{F}$.

Слова для языка LS имеют вид:

t, S, s

Части слова должны иметь определённое содержание, иначе слово отвергается сразу и не принадлежит языку LA как не корректное. В корректном случае содержание этих частей такое:

1.1. Часть t – как и в языке LA – это утверждение, для которого требуется наличие логического доказательства и соблюдение некоторых условий, изложенных далее, чтобы слово принадлежало языку.

По сравнению с языком LA у слова языка LS появились 2 новые части.

1.2. S (заглавная) – параметр числовой длины доказательства, записанный в позиционном (десятичном, например) виде;

1.3. s (прописная) – параметр строковой длины доказательства, размер $|s|$ которого равен S, состоящий только из символов $\overline{1}$. Например, для $S = 10$ корректным будет $s = \overline{1111111111}$. В данном случае для обозначения использована прописная буква потому, что в программировании подобные строковые аргументы и функции часто начинаются с прописной буквы «s».

Логическое доказательство для утверждения t может быть признано доказательством в смысле языка LS лишь в том случае, если предельный (максимальный) размер сертификата, частью которого является доказательство, не превышает полиномиального ограничения от размера $|s|$ и этот предел равен (условимся) $q_S(|s|)$, что равно $q_S(S)$ при корректном слове.

Соотношение $q_S(|s|)$ делает наш язык принадлежащим классу $\mathbb{NP}$, так как помимо полиномиально быстрого относительно размера своих аргументов алгоритма проверки (рассмотренного далее) есть и полиномиальное ограничение размера сертификата относительно размера слова.

1.4. Наличие двух параметров для длины доказательства в слове связано с двумя ограничивающими алгоритмами на время работы алгоритма проверки – для обычных и «специальных» случаев, которые будут построены на базе теоремы о неопределимости – подобно тому, как это было сделано в языке LA.

«Специальными» утверждениями – если сравнивать с 1-й прикладной ролью языка LA – в языке LS будут уже не утверждения вида $AntiSol$, а утверждения вида $AntiSol_S$, так как данное утверждение будет особым уже не в отношении алгоритма-решения Sol(...), но в отношении одновременно алгоритма-решения Sol(..., ..., ...) и параметра числовой длины доказательства S.

2. Про сертификаты для 1-ой прикладной роли языка LS можно сказать всё то же самое, что было сказано про сертификаты в начале раздела «Язык логики LA». Перенесём сюда вид сертификата:

a, y

и уточним, что алгоритм-решение, программный код которого представлен частью a сертификата имеет теперь вид Sol(..., ..., ...), так как алгоритм-решение в обычном случае зависит от аргументов таким образом: Sol(t, S, s).

Алгоритм проверки:

$\mathrm{Ls}(t, S, a, s, y)$ – работает как $\mathrm{Lc}(t, y)$, за исключением того, что:

3.1. В начале работы он проверяет аргументы t, S, a на совпадение с шаблоном:
$\overline{AntiSol_S}, S, \overline{Sol(\dots, \dots, \dots)}$
и если совпадение выявлено, то возвращает 0. Ниже поясню нюансы;

3.2. После 1-го этапа проверяет соответствие между S, s, y. Если соответствия нет, то возвращает 0. А соответствия нет в случаях: Аргумент s состоит не только из символов $\overline{1}$; И/или его длина не равна числу S; И/или размер $|a| + |y|$ превышает полином $1000 \cdot S^n$, например, где n – фиксированное натуральное число);

Кстати, из пункта 1.3 получается в этом случае, что $q_S(|s|) = 1000 \cdot S^n$;

3.3. После 2-го этапа проверяет, чтобы аргумент a соответствовал шаблону: либо $\overline{—}$, либо $\overline{\mathrm{Sol}(..., ..., ...)}$. Если не соответствует, то возвращает 0;

3.4. После этапов 1-3 – если они не закончили работу – переходит к исполнению $\mathrm{Lc}(t, y)$ и возвращает то, что вернёт $\mathrm{Lc}(t, y)$;

Пояснение к пункту 3.1:

Очень важно исполнить этот этап до обращения к потенциально огромным аргументам s и y. Дело в том, что утверждение $\overline{AntiSol_S}$ строится на базе того опровергающего алгоритма $\mathrm{ContrSol}(S)$, который с «правильным» аргументом s, соответствующим алгоритму S, запускает алгоритм:

$\mathrm{Sol}(\overline{AntiSol_S}, S, s)$, где $s = \mathrm{ss}(S)$.

И – важное замечание! – потенциально огромная строка s вовсе не генерируется перед запуском данного алгоритма. Ведь этот запуск – лишь эмуляция работы данного алгоритма и при обращении к любому символу строки s наш опровергающий алгоритм $\mathrm{ContrSol}(S)$;, который эмулирует работу данного алгоритма, возвращает «1» запущенному внутри него алгоритму. И длина строки s в этой эмуляции работы алгоритма имеет длину S. То есть, работа запущенного алгоритма ничем не отличается от случая, в котором у данного алгоритма есть корректный и готовый аргумент s. Но при этом генерировать данную строку не приходится.

Поэтому количество шагов, которое сделает алгоритм $\mathrm{ContrSol}(S)$;, никак не зависит от длины строки s, а зависит (полиномиально) от времени работы запущенного алгоритма (запущенного при как бы уже готовом аргументе s) , зависит (полиномиально) от размера алгоритма $|\mathrm{Sol}(..., ..., ...)|$ и от размера аргумента $|S|$.

И если этот запущенный алгоритм остановится с каким-то результатом, то опровергающий алгоритм $\mathrm{ContrSol}(S)$; выдаст то, что «условно» опровергнет результат работы запущенного алгоритма.

А опровергнет «условно» только потому, что есть ограничения на размер доказательства, а без такого ограничения опровержение было бы буквальным, а именно:

4.1. Если результат будет $\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 1$, то опровергающий алгоритм вернёт 0, что будет соответствовать отрицанию $\overline{AntiSol_S}$ и в этом случае опровержение – буквальное (а не условное) и не зависит от размеров доказательства, так как подходящего доказательства нет вообще.

4.2. Если результат будет $\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 0$, то опровергающий алгоритм вернёт 1, что будет соответствовать утверждению $\overline{AntiSol_S}$ и в этом случае работа алгоритма $\mathrm{Sol}(\overline{AntiSol_S}, S, s) = 0$ будет опровергнута только тогда, когда результат работы опровергающего алгоритма (который действительно имеет место быть) можно доказать доказательством длиной до $q_S(S) - |a|$.

То есть – доказательство состоявшегося факта работы алгоритма заведомо есть, но вопрос только в том, чтобы оно было не слишком длинное.

Но в любом случае алгоритм $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ не может подтвердить доказуемость $\overline{AntiSol_S}$ и об этом сообщает алгоритм проверки

$\mathrm{Ls}(\overline{AntiSol_S}, S, \mathrm{Sol}(..., ..., ...), ..., ...) = 0$

4.3. При этом не имеет значения, какие там аргументы s и y – потому что недоказуемость имеет место быть даже при корректных значениях s и y, а при некорректных значениях – проверка заведомо даст негативный результат.

И этот результат получается очень быстро – неполиномиально быстро относительно размеров аргументов $|s|$ и $|y|$, потому что от них данный результат зависит не больше, чем зависит размер $|S|$ от размера правильного (соответствующего числу S) $|s|$, а это неполиномиально малая (логарифмическая) зависимость.

Построенный алгоритм $\mathrm{Ls}(t, S, a, s, y)$ для языка LS со словами вида:

t, S, s

будем считать первой прикладной ролью языка LS и за счёт этого мы можем отнести язык LS к классу $\mathbb{NP}$. Действительно, алгоритм $\mathrm{Ls}(t, S, a, s, y)$ работает в пределах полиномиального количества шагов относительно размера своих аргументов и размер сертификата (доказательства) полиномиально ограничен относительно размера слова, так как $|y| \leq q_S(|s|) - |a|$.

5.1. А как же получается, что для слова типа t, S, s при утверждении $t = \overline{AntiSol_S}$ и сертификате $a = \overline{\mathrm{Sol}(..., ..., ...)}$ проверка работает неполиномиально быстро относительно размера слова?

Ответ будет неожиданным для «классического» подхода. Проверка, когда она работает неполиномиально быстро в сравнении с «обычной» скоростью, работает вовсе не для слова

t, S, s при утверждении $t = \overline{AntiSol_S}$

и сертификате

$a = \overline{\mathrm{Sol}(..., ..., ...)}, y,$

а для слова:

t, S, a при утверждении $t = \overline{AntiSol_S}$ и $a = \overline{\mathrm{Sol}(..., ..., ...)}$

и сертификатах

$s, y.$

5.2. Дело ведь не в том, каким образом мы выбрали аргументы и в каком порядке расставили их, разделив запятыми. Дело в логике работы алгоритма проверки – он может брать 5 кусков из 1-го аргумента и использовать их как сертификат, а остальное из первого аргумента использовать как слово, добавив к нему ещё частей из второго аргумента. Я для удобства разбил информацию, получаемую алгоритмом проверки на те аргументы, которые написаны.

Но даже по смыслу у нас алгоритм проверки описывает 2 языка. Первый – язык логики Пеано (с ограничениями на размер доказательств), второй – язык доказуемости утверждения не просто в логике Пеано (с ограничениями на размер доказательств), но в логике, сокращённой на некоторые заведомо недоступные доказательства для данного алгоритма-решения.

Можно считать эти 2 языка одним: Где в «первой части» решается вопрос о доказуемости данного утверждения доказательствами допустимого размера «вообще». А во «второй части» – о доказуемости данного утверждения с учётом (помимо размера доказательств) некоторых ограниченных возможностей у заданного алгоритма-решения.

Если для «первой» части у нас такие слово и сертификат из её первой роли:

Слово вида 1: t, S, s; Сертификат вида 1: a, y,

То для «второй части» языка у нас такие слово и сертифика из её первой роли:

Слово вида 2: t, S, a, s; Сертификат вида 2: y.

Но даже это отличается от того, что было выше при утверждении $t = \overline{AntiSol_S}$ и $a = \overline{\mathrm{Sol}(..., ..., ...)}$:

Слово спец. вида: t, S, a; Сертификат спец. вида: s, y.

5.3. Да, отличается, но слово в последнем варианте и не принадлежит языку LS, как не принадлежит ему и никакое другое слово вида 2 при утверждении $t = \overline{AntiSol_S}$ и $a = \overline{\mathrm{Sol}(..., ..., ...)}$:

Слово вида 2: t, S, a, s; Сертификат вида 2: y.

Почему же в «слове спец. вида» часть s «перепрыгнула» в «сертификат спец. вида»? Потому что она даже не рассматривается алгоритмом проверки в данном специальном случае –

при $t = \overline{AntiSol_S}$ и $a = \overline{\mathrm{Sol}(..., ..., ...)}$. Слово оказывается отвергнутым алгоритмом проверки без рассмотрения аргументов s, y.

А что при «классическом» подходе к алгоритму проверки отвергается без рассмотрения? Избыточная часть слишком длинного сертификата. Вот с этой точки зрения часть s в данном случае и оказывается «слишком длинной частью сертификата» - вместе с y.

И вообще – видна условность деления на «часть слова» и «часть сертификата». То, что является в обычном случае частью слова (речь о части s-малое) в специальном случае вообще не рассматривается алгоритмом проверки. И в специальном случае слово отвергается (без рассмотрения s в качестве его части) независимо от того – «корректный» ли $s = \mathrm{ss}(S)$ (это требование для корректности слова в обычном случае), либо часть s содержит вообще «не пойми что». С одной стороны, можно считать, что в специальном случае s – является частью слова, которая отвергается так же, как слишком длинный сертификат – без рассмотрения. А с другой стороны, можно считать, что являющийся в обычном случае частью слова s превращается в специальном случае из части слова в часть сертификата. Разницы как считать – нет, ведь часть s в специальном случае вообще не используется алгоритмом проверки.

5.4. Но, если про часть s можно решить в отношении возможной принадлежности к сертификату и так и эдак когда она не используется, то часть a заведомо используется и оказывается иногда частью слова, а иногда – сертификата. Но ведь аргументы у алгоритма-решения содержат именно информацию о слове, а не сертификате. Как же в этом случае быть с частью a?

А никак. Часть a дана алгоритму-решению не в составе переданных ему аргументов, а в качестве его собственного программного кода.

5.2 Язык LS, пункт 6

6. Для тех, кто решил читать заметку с этого пункта, дадим сводную информацию. Язык LS из класса $\mathbb{NP}$ соответствует следующему алгоритму проверки (по которому язык LS и можно отнести к классу $\mathbb{NP}$) – $\mathrm{Ls}(t, S, a, s, y)$. В неисключительных случаях (об исключительных будет в п. 6.1) аргументы «не подтвердят» слово языка и результат будет 0 (Ноль), если не будет исполнено хоть одно из следующих условий:

S – число, записанное в позиционном виде (например, в десятичном – вроде 3001) :

$s = \mathrm{ss}(S)$, где алгоритм $\mathrm{ss}(S)$ возвращает строку длиной S, состоящую только из символов «1»;

a – текст (программный код) некоторого алгоритма типа $\overline{\mathrm{Sol}(...,...,...)}$ или же текст $\overline{-}$;

t – текст некоторого утверждения на языке теории Пеано;

y – текст некоторого доказательства на языке теории Пеано;

$|y|$ – размер текста y ограничен: $|y| \leq q_S(S) - |a|$, где $q_S(S) = 1000 \cdot S^n$, n – некоторая фиксированная степень, и, разумеется, условие на размер нарушено, если $q_S(S) \leq |a|$;

Утверждение с текстом в t доказывается доказательством с текстом в y в рамках теории Пеано (или любой другой заранее оговорённой теории арифметики, если она более полная и удобная для наших целей, чем теория Пеано);

Если случай не из пункта 6.1 и перечисленные условия выполнены, то:

$\mathrm{Ls}(t, S, a, s, y) = 1$.

Ещё мы условились, что записи вида $\overline{AntiSol_S}$ или $\overline{\mathrm{Sol}(...,...,...)}$ обозначают текст утверждения $AntiSol_S$ на заранее оговоренном языке логики и программный код (тоже текст, фактически) алгоритма $\mathrm{Sol}(...,...,...)$, записанный оговорённым способом представления алгоритмов в выбранной за основу языка LS теории первого порядка (теория Пеано способна представлять алгоритмы, но не очень наглядно – тут есть что улучшать для практических нужд).

И, разумеется, мы пользуемся не «гёделевыми номерами», которые дают неполиномиально огромное представление для информации (число 10, например, имеет вид «1111111111»), а принятым в «человеческом» и «компьютерном» мире представлением в виде текстов и чисел в позиционном (десятичном, например) представлении.

Теперь – о времени работы алгоритма проверки для разных случаев, словах языка LS, сертификатах, их размере и исключениях:

6.1. Для случаев, подходящих под шаблон

$\mathrm{Ls}(\overline{AntiSol_S}, S, \mathrm{Sol}(...,...,...),...,...)$ количество шагов работы – в пределах полинома $p_\exists(|t|, |S|, |a|)$. В силу того, что в данном случае $t = \overline{AntiSol_S}$, чей размер полиномиально зависит от размера $|a| = |\overline{\mathrm{Sol}(...,...,...)}|$ и от размера $|S|$, а при этом $|\overline{\mathrm{Sol}(...,...,...)}| < |\overline{AntiSol_S}|$, то в $p_\exists(|t|, |S|, |a|)$ можно заменить $|a|$ на $|t|$ и ограничивающий алгоритм останется ограничивающим (количество шагов работы алгоритма проверки не превысит результат данного ограничивающего алгоритма) и при этом он останется полиномиальным. И его тогда можно переписать в виде:

$p_\exists(|\overline{AntiSol_S}|, |S|)$ (что равно $p_\exists(|t|, |S|)$).

Проверку под шаблон 6.1 алгоритм $\mathrm{Ls}(t, S, a, s, y)$ проводит в первую очередь и – если соответствие обнаружено – возвращает 0 в пределах указанного времени (количества шагов).

Кстати, многоточия в конце $\mathrm{Ls}(\overline{AntiSol_S}, S, \mathrm{Sol}(...,...,...),...,...)$ стоят потому, что аргументы s, y в данном случае не используются и тут алгоритм зависит, фактически, от трёх аргументов,

возвращая 0: $\mathrm{Ls}(\overline{AntiSol_S}, S, \mathrm{Sol}(...,...,...),...,...) = 0$.

И, напомню, случай 6.1 относится к тому слову (словам!), принадлежность которого отвергается и смысл этого отказа такой: «Алгоритм $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ не может найти доказательство для утверждения $AntiSol_S$». Можно было бы добавить «...в пределах доказательств допустимого размера», но найти доказательство для утверждения $AntiSol_S$ алгоритм $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ не может в принципе. Поэтому не может «вообще» и, в силу этого, «...в пределах доказательств допустимого размера» в том числе.

Раз нет доказательств, значит, нет и соответствующей длины доказательства. А то, что результат (0 – Ноль) алгоритм проверки выдаёт очень быстро, независимо от аргумента s (и от размера $|s|$) – соответствует оптимизации получения самого короткого и быстрого результата из возможных правильных результатов. Так, полиномиальность поиска доказательства рассчитывается от самого короткого из доказательств. И, кстати, тоже может не зависеть от аргумента s – если бы мы включили в алгоритм $\mathrm{Ls}(t, S, a, s, y)$ проверку утверждений на соответствие схеме аксиом из логики предикатов, например.

Вариант из 6.1 – это как 1-я прикладная роль языка из класса $\mathbb{P}$ – алгоритм типа $\mathrm{Lp}(w)$: у этого алгоритма есть «ограничивающий алгоритм» на количество шагов до получение результата (скорость работы) – $p(|w|)$ – полиномиальный от размера $|w|$. Если рядом «положить» значение некоторого сертификата s, от которого ничего не зависит, то ведь всё равно этот сертификат можно дописать в качестве аргумента: $\mathrm{Lp}(w, s)$, хоть он и не используется. Но надо ли после этого считать, что ограничивающий алгоритм стал теперь таким: $p(|w|, |s|)$? Разумеется, нет.

И «нет» потому, что важна логика работы алгоритма, а не форма записи.

Вообще-то спец. случай пункта 6.1 касается принадлежности языку LS слов вида:

t, S, a, s. Соответствующие им сертификаты имеют вид:

y.

Отвергнутые «по шаблону» пункта 6.1 слова имеют вид:

$\overline{AntiSol_S}, S, \overline{Sol(\dots,\dots,\dots)}, s$.

Разумеется часть слова a (которая в данном случае равна $\overline{Sol(\dots,\dots,\dots)}$) не присутствует среди аргументов алгоритма-решения $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$, так как эта информация для алгоритма-решения присутствует в качестве его собственного программного кода. И – в отличие от аргументов – это неизменная и неустранимая данность для данного алгоритма-решения – независимо от того, «понимает» он её или «не понимает».

Но кроме специального случая – при котором соответствующие слова языку LS как раз не принадлежат – в остальном принадлежность слов данного вида ничем не отличается от принадлежности слов из пункта 6.2. Просто часть a в пункте 6.2 уже не является частью слова (а становится частью сертификата). Такая «одинаковость» (кроме спец. случая) связана с тем, что в языке LS мы учитываем только ту ограниченность возможностей алгоритмов-решений, которая следует из теоремы о неопределимости. Но эта ограниченность полностью исчерпывается специфическими случаями (хоть их и бесконечное количество для каждого алгоритма-решения) из пункта 6.1.

6.2. Для остальных случаев количество шагов работы алгоритма проверки – в пределах $p_\forall(|t|, |S|, |s|)$, впрочем, и пункт 6.1 соответствует пункту 6. Но случай 6.1 является более жёстким ограничением – неполиномиально более жёстким, так как в случае 6.2 членом полинома

является размер $|s|$, растущий экспоненциально относительно размера $|S|$.

Случай 6.2. относится к словам вида:

t, S, s.

Эти слова не учитывают алгоритм-решение, который ищет доказательство утверждения t и не содержат эту часть (a) в составе слова.

Но эти слова из случая 6.2 зачастую принадлежат языку LS, то есть, имеют соответствующий подтверждающий сертификат вида:

a, y.

И этот сертификат включает в себя часть a.

При этом его суммарный размер ограничен (требование корректности) алгоритмом

$1000 \cdot S^n$, где n – некоторая фиксированная степень:

$|a| + |y| \leq 1000 \cdot S^n$, что повторяет сказанное в начале пункта 6 про размер доказательства $|y|$:

$|y| \leq q_S(|s|) - |a|$

Для 1-й прикладной роли языка из класса $\mathbb{NP}$ алгоритм проверки должен работать полиномиальное время относительно размера слова. Но при этом алгоритм проверки зависит и от сертификата. А из-за этого и сертификат – в своей значимой части – должен по своему размеру быть в подобных же полиномиальных пределах относительно размера слова.

Именно поэтому у ограничивающих полиномиальных алгоритмов на время работы проверки и на размер сертификата зачастую стоят одни и те же размерные аргументы. Хотя у ограничивающего алгоритма на размер может быть и нечто меньшее (сколь угодно) – условия на полиномиальность всё равно будут исполнены, пункт 6.1 тому пример.

6.3. Если алгоритм-решение $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ для слова из специального случая 6.1 возвращает 1 при корректном $s = \mathrm{ss}(S)$:

$\mathrm{Sol}(\overline{AntiSol_S}, S, s) = 1$ (случай 1),

то доказательства для утверждения $AntiSol_S$ нет, и, напротив, имеется доказательство для его отрицания. То есть, ответ данного алгоритма-решения на данном слове в этом случае заведомо не корректен.

Если же алгоритм-решение $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ для слова из специального случая 6.1 возвращает 0 при корректном $s = \mathrm{ss}(S)$:

$\mathrm{Sol}(\overline{AntiSol_S}, S, s) = 0$ (случай 2),

то логическое доказательства для утверждения $AntiSol_S$ имеется, но, вопрос – соответствует ли размер данного доказательства допустимым размерам:

$|y| \leq q_S(|s|) - |a|$.

Но если размер данного доказательства соответствует указанному пределу, то ответ данного алгоритма-решения на данном слове и в этом случае оказывается не корректным.

Если же алгоритм-решение $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ для слова из специального случая 6.1 не возвращает никакого ответа при корректном $s = \mathrm{ss}(S)$ (случай 3), то данное слово действительно не принадлежит языку LS и это подтверждается при любых сертификатах алгоритмом проверки. То есть, и в данном случае работа алгоритма-решения $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ оказывается не корректной.

Значит, единственный случай, при котором алгоритм-решение $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ может выдать корректный результат – случай 2. И результат этот должен быть 0. И этот результат

соответствует результату работы алгоритма проверки $\mathrm{Ls}(\overline{AntiSol_S}, S, \mathrm{Sol}(...,...,...),...,...) = 0$ за полиномиальное количество шагов работы – в пределах $p_{\exists}(|\overline{AntiSol_S}|, |S|)$ (что равно $p_{\exists}(|t|, |S|)$).

Поэтому при сведении нашей задачи из класса $\mathbb{NP}$ в класс $\mathbb{P}$ полиномиальность времени проверки для случая 2 $p_{\exists}(|t|, |S|)$ должна превратиться в полиномиальность времени на обнаружение решения (или невозможности решения) вида $p_{\exists+}(|t|, |S|)$.

И вот идея доказательства для $\mathbb{NP} \neq \mathbb{P}$:

Здесь очевидно, что при достаточно больших S данное предельное время работы алгоритма-решения из случая 2 экспоненциально мало по сравнению с допустимым размером доказательств: $|y| \leq q_S(|s|) - |a|$, так как размер $|s| = |\,\mathrm{ss}(S)|$ экспоненциально велик по сравнению и с размером $|a|$, и с аргументами в $p_{\exists+}(|t|, |S|)$ при достаточно больших S.

А это значит, что для достаточно больших S среди допустимых доказательств найдётся и то, в котором будет вывод и для случая 2: $\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 0$, и для вытекающего из него утверждения $AntiSol_S$. А это и будет означать, что алгоритм $\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S))$ не смог подтвердить истинность утверждения $AntiSol_S$ для которого есть доказательство среди доказательств допустимого размера.

А поскольку алгоритм $\mathrm{Sol}(...,...,...)$ произвольный, то это будет означать отсутствие корректных алгоритмов-решений, делающих язык LS языком класса $\mathbb{P}$. То есть – оказывается доказанным неравенство $\mathbb{NP} \neq \mathbb{P}$.

Конечно, в нескольких абзацах выше изложено доказательство $\mathbb{NP} \neq \mathbb{P}$ в общих чертах, а детали – как строить вывод для утверждения $AntiSol_S$ при корректном времени работы алгоритма-решения $\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 0$ и почему размер вывода будет допустимого размера – изложены в следующем разделе.

6 $\mathbb{NP} \neq \mathbb{P}$ на примере языка LS.

При сведении нашей задачи из класса $\mathbb{NP}$ в класс $\mathbb{P}$ полиномиальность времени проверки $p_\exists(|t|,|S|)$ и $p_\forall(|t|,|S|,|s|)$ из пп. 6.1 и 6.2 соответственно

должны превратиться в полиномиальность времени на обнаружение решения (или невозможности решения) вида:

$p_{\exists+}(|t|,|S|)$ и $p_{\forall+}(|t|,|S|,|s|)$ соответственно

Но нас будет интересовать специфический случай, с его $p_{\exists+}(|t|,|S|)$, так как проблема на пути поиска решения за данное время тут особенно очевидна:

Есть все основания полагать, что данное небольшое время работы приведёт к противоречию с тем, что после ответа алгоритма-решения при достаточно большом S возникнет доказательство для отвергнутого алгоритмом-решением утверждения (в соответствии с пунктом 6.3) и доказательство это будет достаточно коротким по меркам пункта 6.2, чтобы вписаться в размеры, диктуемые размером $|s|$.

При этом единственный случай, когда алгоритм $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ может выдать корректный ответ такой:

$\mathrm{Sol}(\overline{AntiSol_S}, S, s) = 0$ (случай 2 из пункта 6.3 предыдущего раздела).

Если при этом для утверждения $AntiSol_S$ найдётся доказательство y в пределах из пункта 6.2 прошлого раздела:

$|y| \leq q_S(|s|) - |a|$,

то алгоритм-решение $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ окажется некорректным – сообщил об отсутствии доказательства утверждения $AntiSol_S$ в допустимых пределах, а доказательство есть и размер его – допустимого размера.

С точки зрения языка LA, где нет ограничений на размер доказательства, любой алгоритм-решение заведомо некорректен, так как случай 6.1 из предыдущего раздела (если пункт 6 переписать для LA) сообщает о невозможности для алгоритма-решения $Sol(\overline{AntiSol})$ (произвольного) найти доказательство для $AntiSol$, но ответ алгоритма-решения «не нашёл доказательство» создаёт это доказательство в соответствии с пунктом 6.3, случай 2 предыдущего раздела.

Отличие языка LS от языка LA в том, что при $\mathrm{Sol}(\overline{AntiSol_S}, S, ...) = 0$ альтернатива данному ответу – доказательство утверждения $AntiSol_S$ – может и не уложиться в допустимые пределы $q_S(S)$.

То есть, для того, чтобы свести нашу 1-ю прикладную роль языка LS класса $\mathbb{NP}$ к 1-й прикладной роли класса $\mathbb{P}$ нам нужно найти соответствующий корректный алгоритм-решение $\mathrm{Sol}(t, S, s)$, который в случае пункта 6.1 из прошлого раздела успевает выдать результат 0 за время (количество шагов) в пределах:

$N_{\exists+} \leq p_{\exists+}(|\overline{AntiSol_S}|, |S|)$ (или короче – $N_{\exists+} \leq p_{\exists+}(|t|,|S|)$.

И при этом алгоритм-решение $\mathrm{Sol}(t, S, ...)$ не использует (не обращается) к аргументу s-прописная в большей степени, чем это небольшое количество шагов.

Но помимо этого, в случае 6.1 частью слова является a, которого мы не видим среди аргументов $\mathrm{Sol}(t, S, s)$. Его нет среди аргументов потому, что данная информация имеется у алгоритма $\mathrm{Sol}(..., ..., ...)$ в качестве его собственного программного кода. И эту информацию алгоритм может получить при помощи диагонализации, как известно (не термин, а факт) со времён Гёделя.

Способность $\mathrm{Sol}(..., ..., ...)$ в решении вопроса:

$\mathrm{Sol}(\overline{AntiSol_S}, S, s)$, где $s \neq \mathrm{ss}(S)$

мы не рассматриваем – в этих случаях слово явно не принадлежит языку LS и алгоритму Sol(..., ..., ...) не мешает ничего (кроме, может быть, его собственного устройства) дать правильный ответ – 0 (Ноль). Нас интересует только вариант:

$\mathrm{Sol}(\overline{AntiSol_S}, S, s)$, где $s = \mathrm{ss}(S)$

Итак, чтобы доказать некорректность алгоритма-решения $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ будем исходить из того, что он возвращает 0: $\mathrm{Sol}(\overline{AntiSol_S}, S, s) = 0$ за количество шагов $N_{\exists+} \leq p_{\exists+}(|\overline{AntiSol_S}|, |S|)$ (или короче – $N_{\exists+} \leq p_{\exists+}(|t|, |S|)$). При этом $s = \mathrm{ss}(S)$. И нам нужно доказать, что для достаточно большого S-заглавного можно при этом доказать утверждение $AntiSol_S$ доказательством с размером $Q_{\exists+} = |y|$ таким, что:

$Q_{\exists+} \leq q_S(|s|) - |a|$.

Вот и будем соответствующее доказательство строить в несколько этапов...

1.1. Для разбираемого случая

$\mathrm{Sol}(\overline{AntiSol_S}, S, s) = 0$

у нас есть истинное (доказуемое за фиксированное число шагов) утверждение для произвольного S:

$(\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 0) \Rightarrow AntiSol_S$.

Подобные импликации крайне просто доказываются для алгоритмов - благодаря аксиоме равенства J_2.

У нас есть простое верное равенство для произвольного S:

$\mathrm{ContrSol}(S) = \mathrm{If}(\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 0, 1, 0)$

Это равенство описывает, как работает алгоритм ContrSol(S), лежащий в основе $AntiSol_S$ и «поступающий наоборот». Алгоритм

If(Условие, Результат при истине, Результат в ином случае)

широко известен в программировании. Если обозначить приведенное равенство как $A(i)$, где

i обозначает $\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S))$, и пусть

j обозначает 0 (Ноль), тогда

Воспользуемся аксиомой J_2:

$(i = j) \Rightarrow (A(i) \Rightarrow A(j))$

Переписав его с учётом истинности A(i) в виде:

$(i = j) \Rightarrow A(j)$, и получим подстановкой:

$(\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 0) \Rightarrow (\mathrm{ContrSol}(S) = \mathrm{If}(0 = 0, 1, 0))$

А с учётом свойств алгоритма If(...,...,...) получим:

$(\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 0) \Rightarrow (\mathrm{ContrSol}(S) = 1)$

В силу того, что равенство $\mathrm{ContrSol}(S) = 1$ и есть утверждение $AntiSol_S$ – приходим к тому, что и требовалось доказать:

$(\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 0) \Rightarrow AntiSol_S$.

Какого размера будет этот этап доказательства с учётом необходимой подстановки конкретного значения S? С учётом того, что строки доказательства пункта 1.1 – производные от $\overline{Sol(\ldots, \ldots, \ldots)}$ и числа S и количество строк фиксировано – будет полином от размеров тех же $|\overline{Sol(\ldots, \ldots, \ldots)}|$ и размера $|S|$. Но для единообразия вместо $|\overline{Sol(\ldots, \ldots, \ldots)}|$ используем $|t| = |\overline{AntiSol_S}|$, что приведёт к завышению полинома, но тем более он будет давать большую

величину, чем наибольший размер нашего доказательства для пункта 1.1. И получим полином q_1 такого вида:

$q_1(|t|, |S|)$

Заметим, что обсуждаемая импликация истинна и её можно вообще сделать схемой аксиом для произвольного алгоритма-решения $\mathrm{Sol}(..., ..., ...)$ и такое расширение теории Пеано будет вполне корректным вариантом теории первого порядка для арифметики. И уж в этом случае оценка размера доказательства конкретной реализации данной схемы аксиом очевидно соответствует последнему ограничивающему алгоритму.

1.2. Осталось решить, какой длины можно построить доказательство для

$\mathrm{Sol}(\overline{AntiSol_S}, S, s) = 0$ при $s = \mathrm{ss}(S)$

Если бы алгоритм был без аргументов, то для такого варианта:

$\mathrm{SolSpec_S}() = 0$ и количестве шагов его работы N_{Spec}

длина доказательства была бы в пределах полинома:

$q_2(|\overline{\mathrm{SolSpec_S}()}|, N_{Spec})$

что довольно очевидно, так как каждый шаг работы алгоритма потребует полиномиального количества шагов логического вывода, которые представляют этот шаг и размер которых зависит от размера алгоритма. Степени этого полинома нас тут не интересуют, но важно, что зависимость полиномиальная.

Как свести алгоритм с аргументами к алгоритму без аргументов? Самый простой путь: составной алгоритм, в котором в первой части нужные значения аргументов присваиваются переменным, а затем во второй части вызывается исходный алгоритм с данными переменными в качестве аргументов. Тогда условный программный текст (где знак равенства – это присваивание, многоточие – нужные числа, а результат работы программы – это результат алгоритма в последней строке программного кода) для $\mathrm{SolSpec_S}()$ на базе $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ был бы таким:

$t = \overline{AntiSol_S}$;

$S = \ldots$;

$s = \mathrm{ss}(S)$;

$\mathrm{Sol}(t, S, s)$;

Но в этой программе нас не устраивает строка $s = \mathrm{ss}(S)$. Она может работать неполиномиально долго по сравнению со всей остальной программой. Поэтому в отношении операций с аргументом s-прописная нам надо переделать алгоритм $\mathrm{Sol}(t, S, s)$ на такой, в котором каждое обращение к любому i-му символу аргумента s-прописная заменялось бы на обращение к функции:

$Func_S(i)$

и эта функция эмулировала бы аргумент-строку s-прописная, возвращая:

Символ $\overline{1}$, если $1 \leq i \leq S$

или информацию о выходе запроса за пределы данных, если $i < 1$ или $i > S$.

Очевидно, что такую модернизацию алгоритма $\mathrm{Sol}(t, S, s)$ в некий алгоритм

$\mathrm{SolX}(t, S)$

можно осуществить и довольно легко. В название алгоритма добавлен символ «Икс», что часто делают для «улучшенных» версий алгоритмов или данных – вроде файлов типа «.docx» вместо прежних «.doc», например – этот «Икс» - кусочек от слова «Extention».

И вот с учётом такой модернизации наша программа для $\mathrm{SolSpec_S}()$ примет вид:

$t = \overline{AntiSol_S}$;

$S = \ldots$;

$\mathrm{SolX}(t, S)$;

И очевидно, что верно:

$\mathrm{SolSpec_S}() = \mathrm{Sol}(\overline{AntiSol_S}, S, s)$ при $s = \mathrm{ss}(S)$

То есть, для последнего равенства есть доказательство или его можно и вовсе аксиоматизировать, в некую схему аксиом.

Размер доказательства для последнего равенства – аналогично прошлому пункту 1.1 можно оценить (не больше чем) так:

$q_2(|t|, |S|)$

1.3. Последнее равенство позволяет нам свести вопрос о работе алгоритма:

$\mathrm{Sol}(\overline{AntiSol_S}, S, s) = 0$ при $s = \mathrm{ss}(S)$

К работе алгоритма

$\mathrm{SolSpec_S}() = 0$

Но это уже позволяет оценить размер доказательства, который доказывает последнее равенство:

$q_3(|\overline{\mathrm{SolSpec_S}()}|, N_{Spec})$, где N_{Spec} – количестве шагов работы $\mathrm{SolSpec_S}()$.

Если посмотреть на программу, то там всё - производное от программного кода алгоритма $\mathrm{Sol}(..., ..., ...)$ и числа S. При этом размеры этих производных данных полиномиально зависят от размеров $|\,\mathrm{Sol}(..., ..., ...)|$ и $|S|$ (аргумент s-прописная, который зависит неполиномиально, мы в последней версии исключили), поэтому последний ограничивающий алгоритм можно переписать так:

$q_4(|\overline{\mathrm{Sol}(..., ..., ...)}|, |S|, N_{Spec})$, где N_{Spec} – количестве шагов работы $\mathrm{SolSpec_S}()$.

Еще и количество шагов работы программы $\mathrm{SolSpec_S}()$ тоже полиномиально зависит от количества $N_{\exists+}$ шагов работы нашего исходного алгоритма-решения, от размеров исходного алгоритма $|\,\mathrm{Sol}(..., ..., ...)|$, размеров $|S|$, размеров $|\overline{\mathrm{Func_S}(...)}|$ – с учётом дополнительных строк в $\mathrm{SolSpec_S}()$ в сравнении с исходным $\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S))$ и заменой части кода и части шагов выполнения в самой $\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S))$ на операции с функцией $\mathrm{Func_S}(i)$ (модернизация до $\mathrm{SolX}(t, S)$).

То есть, найдётся некоторый полином, для которого верно:

$N_{Spec} \leq p_1(|\overline{\mathrm{Sol}(..., ..., ...)}|, |\overline{\mathrm{Func_S}(...)}|, |S|, N_{\exists+})$

С учётом того, что $|\overline{\mathrm{Func_S}(...)}|$ меняется (полиномиально по размерам) только от |S|, то его можно оставить как часть зависимости полинома от |S|, но исключив, как отдельный аргумент. Тогда последнюю оценку можно переписать так:

$N_{Spec} \leq p_2(|\overline{\mathrm{Sol}(..., ..., ...)}|, |S|, N_{\exists+})$

Тогда:

$q_4(|\overline{\mathrm{Sol}(..., ..., ...)}|, |S|, N_{Spec}) \leq q_4(|\overline{\mathrm{Sol}(..., ..., ...)}|, |S|, p_2(|\overline{\mathrm{Sol}(..., ..., ...)}|, |S|, N_{\exists+}))$

Композиция полиномов даёт полином с теми же аргументами, что у всех полиномов композиции. Поэтому имеем:

$q_4(|\overline{\mathrm{Sol}(..., ..., ...)}|, |S|, N_{Spec}) \leq q_5(|\overline{\mathrm{Sol}(..., ..., ...)}|, |S|, N_{\exists+})$

Поскольку $|AntiSol_S| \geq |\overline{\mathrm{Sol}(..., ..., ...)}|$, а $|AntiSol_S|$ обозначаем тут как $|t|$,

то можно заменить в последнем неравенстве правую часть так:

$q_4(|\overline{\mathrm{Sol}(..., ..., ...)}|, |S|, N_{Spec}) \leq q_5(|t|, |S|, N_{\exists+})$

Значит, размер для доказательство
$\mathrm{SolSpec_S}() = 0$
будет в пределах
$q_5(|t|, |S|, N_{\exists+})$
Но мы знаем, что
$N_{\exists+} \leq p_{\exists+}(|t|, |S|)$
И это даёт такое ограничение при подстановке:
$q_5(|t|, |S|, p_{\exists+}(|t|, |S|))$
И опять, раскрывая композицию полиномов, получаем верхний предел для размера доказательства для равенства
$\mathrm{SolSpec_S}() = 0$
такой:
$q_6(|t|, |S|)$

1.4. Теперь у нас есть полная информация о построении доказательства утверждения $AntiSol_S$ для случая 2 пункта 6.3 предыдущего раздела и можно выяснить, в каких пределах находится размер данного доказательства:

1. Доказываем $\mathrm{SolSpec_S}() = 0$ из пункта 1.3 с размером этой части доказательства не более чем $q_6(|t|, |S|)$ символов;

2. Доказываем $\mathrm{SolSpec_S}() = \mathrm{Sol}(\overline{AntiSol_S}, S, s)$ при $s = \mathrm{ss}(S)$ из пункта 1.2 с размером этой части доказательства не более чем $q_2(|t|, |S|)$ символов;

3. Из пунктов 1 и 2 выводим по аксиомам равенства $\mathrm{Sol}(\overline{AntiSol_S}, S, s) = 0$, о размере скажем после пунктов о доказательстве;

4. Доказываем $(\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S)) = 0) \Rightarrow AntiSol_S$ из пункта 1.1 с размером этой части доказательства не более чем $q_1(|t|, |S|)$ символов;

5. Из пунктов 3 и 4 выводим по правилу MP искомое утверждение $AntiSol_S$, о размере скажем после пунктов о доказательстве;

Таким образом, доказательство утверждения $AntiSol_S$ построено. Если считать размер доказательства в этапах (очень простых) 3 и 5 вместе не превышающим следующий полином:
$q_7(|t|, |S|)$
То суммарный размер $Q_{\exists+}$ для доказательства утверждения $AntiSol_S$ не превысит некого полинома $q_{\exists+}(|t|, |S|) = q_1(|t|, |S|) + q_2(|t|, |S|) + q_6(|t|, |S|) + q_7(|t|, |S|)$:
$Q_{\exists+} \leq q_{\exists+}(|t|, |S|)$

Сравниваем этот ограничивающий алгоритм с ограничивающим алгоритмом на размер доказательства из пункта 6.2 предыдущего раздела:
$|y| \leq q_S(|s|) - |a|$
Очевидно, что для достаточно больших S верно:
$q_{\exists+}(|t|, |S|) + |a| \leq q_S(|s|)$
так как размер $|s|$ экспоненциально велик относительно размеров $|t|$, $|S|$, $|a|$ при достаточно больших S. А степень полинома $q_S(|s|)$ не имеет значения, когда под знаком степени – экспоненциально растущая величина $|\mathrm{ss}(S)|$: такой полином превысит любой другой полином $q_{\exists+}(|t|, |S|)+|a|$, если у другого полинома под степенью любое значение экспоненциально отстаёт в своём росте от значения $|\mathrm{ss}(S)|$.

Таким образом, мы доказали, что для достаточно больших S и случай 2 пункта 6.3 предыдущего раздела оказывается невозможным для корректного решения алгоритмом-решением (произвольным) Sol(..., ..., ...). А это означает, что свести язык LS из класса сложности $\mathbb{NP}$ к классу сложности $\mathbb{P}$ невозможно. То есть, доказано:

$\mathbb{NP} \neq \mathbb{P}$.

7 О «классическом» доказательстве $\mathbb{NP} \neq \mathbb{P}$ (эвристически).

Соберу в этом заключительном разделе те сопутствующие вопросы и ответы, которые возникли по ходу написания данной статьи и показались мне заслуживающими упоминания.

1. «Забота» о псевдоинвалидах

В классическом случае у нас только один набор для скорости работы проверки и размера доказательства, а не 2, как в разобранном случае. Но язык с 2-мя наборами «скорость проверки» + «размер доказательства» (даже с 3-мя, но мы воспользуемся 2-мя) построен. Этот построенный язык LS принадлежит классу $\mathbb{NP}$ и в качестве данных для этого языка от алгоритма-решения требуется использовать собственный программный код. Этот код – есть.

И если «со стороны» можно делать расчёты на основании данного кода, то того же можно требовать от корректного алгоритма-решения – для получения собственного программного кода у алгоритма есть метод диагонализации. Или в отношении требования получать свой программный код надо проявить снисхождение к алгоритму-решению? Но это тогда не математика, а «забота» о псевдоинвалидах. И надо тогда заодно прекратить требовать от алгоритмов-решений умения пользоваться собственными аргументами. Тогда в такой «псевдосоциальной» математике сразу ясно, что $\mathbb{NP} \neq \mathbb{P}$, потому что чего же взять с (якобы) «инвалидов»...

2. Если алгоритм-решение переберёт все доказательства (допустимого размера)

Если бы $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ мог за время $p_{\exists+}(|\overline{AntiSol_S}|, |S|)$ перебрать все доказательства размером до $q_S(|s|) - |a|$, то доказательств для утверждения $AntiSol_S$ не осталось бы.

Тут надо пояснить, что «перебрать» здесь означает поиск нужного доказательства и – если бы оно было обнаружено – завершение работы с результатом 1. Для такого перебора можно рассмотреть и метод прямого перебора всех доказательств допустимой длины – невероятно долгий (экспоненциально!) но вполне тривиальный, но вдруг есть какой-то его полиномиальный аналог.

Но из-за того, что алгоритм $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ в принципе не может выдать корректный результат 1 про утверждение $AntiSol_S$, получается, что все доказательства, которые он «переберёт» с «готовностью» выдать 1, будут им «испорчены». Вот поэтому и получается сказанное выше «доказательств для утверждения $AntiSol_S$ не осталось бы».

Но это заведомо невозможно для больших S за время $p_{\exists+}(|\overline{AntiSol_S}|, |S|)$ из-за неполиномиального роста $q_S(|s|) - |a|$ относительно роста $p_{\exists+}(|\overline{AntiSol_S}|, |S|)$.

3. Док-во $\mathbb{NP} \neq \mathbb{P}$ в «классическом» варианте (эвристически)

Хотя сам классический подход в контексте вопроса об $AntiSol_S$ при алгоритме-решении $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ кажется надуманным, но даже если «разрешить» алгоритму $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ отвечать неполиномиально долго в сравнении со скоростью доступа к нужной информации, то есть не в пределах ограничивающего полиномиального алгоритма вида $p_{\exists+}(|t|, |S|)$, а с «обычным» для остальных утверждений ограничивающим алгоритмом $p_{\forall+}(|t|, |S|, |s|)$, то что принципиального измениться?

Это ведь всё равно не полный перебор, упомянутый в пункте 2 выше, а всего лишь полином, малость которого на фоне неполиномиально огромного количества возможных доказательств куда существеннее, чем разница между самими 2-мя ограничивающими алгоритмами $p_{\exists+}(|t|, |S|)$ и $p_{\forall+}(|t|, |S|, |s|)$.

Притом про утверждение $AntiSol_S$ заранее известно, что алгоритм $\mathrm{Sol}(\overline{AntiSol_S}, S, s)$ не мо-

жет его корректно ни подтвердить (в принципе), ни отвергнуть (а вот здесь условно – если не ограничиваться размерами доказательств). Да и сам анализ трудности решения задачи с опорой на ограниченные возможности данного (произвольного) алгоритма-решения представляется продуктивным, и проведённое доказательство для «не классического» языка из класса $\mathbb{NP}$ с утверждением $AntiSol_S$ может в этом отношении оказаться полезным и при «классическом» подходе.

Если бы удалось найти допустимое по размеру доказательство для результата работы алгоритма-решения, то вопрос о его неспособности был бы решён – примерно так, как в доказательстве из прошлого раздела. Но время (количество шагов) корректного алгоритма-решения полиномиально велико относительно допустимого размера доказательств. Прямой «перевод» работы алгоритма в доказательство можно сделать линейным в смысле превращения времени работы в размер доказательства: Это аналог протоколов работы программ, которые используются программистами. Но такой «протокол» будет явно больше допустимого размера доказательства.

Однако – есть сильные способы сокращения «протокола работы»:

Ведь суть алгоритма – в повторяемости действий. И чем дольше время его работы, тем больше количество и размер отрезков исполнения, которые повторяют себя и могут быть представлены неким производным логическим выводом каких-то результатов из неких «начальных условий».

Если разрешить в нашем языке выстраивать производные правила вывода и сокращения для записи строк вывода, то делать это можно будет неполиномиальным количеством способов от предельной длины доказательств. И чтобы подобрать из всех возможных доказательств то, где эти производные правила и сокращения оптимальны – можно использовать алгоритмы, работающие неполиномиально долго: Интересен только результат – короткое доказательство о результате работы алгоритма-решения.

А вот у корректного алгоритма-решения возможности работать неполиномиально долго относительно предельного размера доказательства – нет. И из-за этого у него есть все «шансы» проиграть в обнаружении наиболее оптимальных способов своей работы и использовать их не хуже, чем они представлены в некоторых допустимых по своим размерам доказательствах.

Но, с другой стороны, время работы алгоритма-решения в фиксированной степени больше, чем предельный размер доказательств. И чтобы скомпенсировать это колоссальное различие, с подбором подходящих производных правил и сокращений надо очень постараться.

Да и с производными правилами вывода надо следить, чтобы время работы алгоритма проверки с ними не превышало заранее заданный ограничивающий полином.

В общем – если пытаться дать доказательство в рамках «классического» подхода к задачам из $\mathbb{NP}$ (с единственным ограничивающим полиномом на время работы у алгоритма проверки), то мне не хватает в настоящий момент ни времени, ни знаний, ни сил для решения данного вопроса. И следуя примеру «разумного» алгоритма-решения Sol(..., ..., ...) в отношении утверждения $AntiSol_S$, я «выдаю ноль» в качестве результата своих недолгих попыток в этом направлении. А другие «алгоритмы» (люди и коллективы) при желании могут опереться на мои завершенные результаты в своей работе.

4. Значимость персонализации

В связи с шуточным сравнением себя с «разумным» алгоритмом Sol(..., ..., ...) в последнем абзаце предыдущего пункта можно снова вернуться к вопросу о рефлексии, которого мы ка-

сались в разделе «Рефлексия на языке LA. $\mathbb{NF} \neq \mathbb{F}$». Интересно, что оперативный ответ алгоритма $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ о невозможности ему подтвердить истинность $AntiSol_S$ приводит к появлению доказательства в тех же полиномиальных рамках, что и скорость ответа – то есть с размером в пределах $p_{\exists+}(|t|, |S|)$, а не $q_S(|s|)$ - |a|.

При этом оперативный ответ означает в случае результата 0 как раз то, что доказательство есть! И иначе ответить «доказательство есть» за время $p_{\exists+}(|t|, |S|)$ алгоритму $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ невозможно при достаточно больших S.

И при достаточно больших S оперативный ответ от $\mathrm{Sol}(\overline{AntiSol_S}, S, ...) = 0$ (при произвольном s) позволяет другим алгоритмам убедиться в истинности утверждения $AntiSol_S$ без неполиномиально огромных задержек, которые возникают, если $\mathrm{Sol}(\overline{AntiSol_S}, S, \mathrm{ss}(S))$ «тянет время».

Заметим, что значения «1» и «0» мы условились использовать в качестве обозначения смыслов: «утверждение принадлежит языку LS в соответствии с полученными данными» и «принадлежность утверждения языку LS не подтверждаются полученными данными» соответственно. Но в случае с оперативным ответом $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ об утверждении $AntiSol_S$ смысл обусловлен не нами, а обстоятельствами. И результаты «1» и «0» меняют свой смысл на противоположный «нашему», если угодно.

И этот «особенный» смысл является лишь «полуфабрикатом» для того, чтобы другие алгоритмы дали уже правильный ответ в соответствии с обычными «договоренностями» о смысле «1» и «0». В данном случае работа корректного алгоритма $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ выступает лишь как часть общей работы некого «коллективного алгоритма». В соответствии с тезисом Чёрча и этому «коллективному алгоритму» можно сопоставить некий вполне конкретный алгоритм, но тут мы видим как на формальном уровне проявляется «часть» и «целое», «индивидуальное» и «коллективное». Определенная работа алгоритма-решения $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$ не может иметь смысла вне рамок более общего результата.

Нечто аналогичное мы имеем и в человеческих коллективах – человек не должен считать себя Наполеоном и пытаться брать на себя задачи, которые ему не по силам, внося хаос и задержку в функционирование «коллективного алгоритма». И умение выдать отрицательный результат - «я не справлюсь с этим в такие сроки» - важная способность к рефлексии и «персонализации» задачи применительно к себе. Без такой способности к корректной «персонализации» твоя деятельность внутри коллектива оказывается деструктивной, а ты становишься вредным элементом, который требуется исключить из коллектива для нормализации его работы.

И ещё кое-что интересное: На примере смены смыслов 1 и 0 местами для некоторых случаев видна «этапность» нарастания сложности – тут нам приходится отказаться от тотальной однозначности понимания (меняя местами смысл 1 и 0), там и сям – ещё какие-то «исключения» и в итоге – неразрешимый вопрос даже в отношении смыслов, который имеют 1 и 0 при ответе разных алгоритмов-решений.

5. О $\mathbb{NP}$-полной задаче ДНФ

Известна теорема о наличии $\mathbb{NP}$-полных задач, одной из которых (первая найденная и самая известная из них) является ДНФ – всяческие дизъюнктивные нормальные формы из логики высказываний.

Из того, что у нас имеется ДНФ для любой единичной задачи для любого языка из класса

сложности $\mathbb{NP}$, которая разрешима, вообще говоря, для алгоритмов – создаётся иллюзия, что дело именно в решении ДНФ. Но эта ДНФ – если почитать доказательство о $\mathbb{NP}$-полноте данной задачи – строится с учётом полиномиальной ограниченности времени работы алгоритма-решения и, фактически, тут нет возможности применять произвольный алгоритм-решение – набор возможностей ограничен. А для большего времени работы при другой единичной задаче строится уже другая ДНФ.

Дело не в отдельных ДНФ, а в их комплексе, а вот их комплекс как раз и может составлять то утверждение $AntiSol_S$, которое неразрешимо для соответствующего ему (взятого произвольно!) алгоритма-решения $\mathrm{Sol}(\overline{AntiSol_S}, S, ...)$.

Поэтому «сводимость» к ДНФ – это лишь иллюзия упрощения, как мне представляется. Для того, чтобы доказать несводимость к языку класса $\mathbb{P}$ языка LS из класса $\mathbb{NP}$, нам всё равно пришлось рассмотреть алгоритмы и воспользоваться теоремой о неопределимости – то есть решать вопрос о комплексе конкретных ДНФ, если рассматривать доказательство $\mathbb{NP} \neq \mathbb{P}$ через призму соответствующей части $\mathbb{NP}$-полного языка ДНФ.

А оперируя с алгоритмами (а не ДНФ), нам потребовалось лишь сделать кое-какие не слишком сложные построения. Видимо, это гораздо более простой путь доказательства $\mathbb{NP} \neq \mathbb{P}$, чем пытаться решать вопрос о несводимости ДНФ к классу $\mathbb{P}$ «в лоб» с учётом того, что решает вопрос о истинности конкретного ДНФ не алгоритм-решение, а какая-то урезанная версия этого алгоритма, да ещё и по-разному урезанная для разных конкретных ДНФ, но при этом подчиняться все эти урезанные версии алгоритма-решения должны общему (полиномиальному) способу «урезания» для всего комплекса единичных задач в этой части из $\mathbb{NP}$-полной задачи.

Список литературы

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

[2] *Д. Гильберт, П. Бернайс.* Основания математики. Логические. исчисления и формализация арифметики Наука, М. 1979.

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