МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ярославский государственный университет им. П.Г.Демидова»

Кафедра компьютерной безопасности и математических методов обработки информации

Сдано на кафедру
«_____» _____ 20_____ г.
Заведующий кфедрой
д.ф.-м.н., профессор
_____ Дурнев В.Г.

Выпускная квалификационная работа

Алгоритм Тарского: изучение и исследование, программная реализация и приложения

специальность 10.05.01 Компьютерная безопасность

Научный руководитель
профессор, д.ф.-м.н.,
_____ Дурнев В.Г.
«_____» _____ 20_____ г.

Студент группы КБ-61СО
_____ Гибадулин Р.А.
«_____» _____ 20_____ г.

Ярославль 2022 г.

Реферат

Данная работа содержит 23 страницУУУУУУ, в работе использовано ?????? источников.

!!! Нужно ли меня нумеровать !!!

!!!!!!!!!!!!!!!

Оглавление

Введение

!!!Во-первых, про приложения алгоритма пока ничего нет, мб стоит выпилить из введения

!!!Во-вторых, написано про реалзиацию алгоритма, не указано, что я сделал только частный случай

!!!В-третьих, перечитать описание каждой из глав

!!!В-четвертых, прочитать рекомендации — https://edunews.ru/students/vypusknaya/kak-pisat-vvedenie-dlya-diploma.html

Пусть $\mathcal{A}$ — формула логики высказываний. Сформулируем следующую задачу: определить, является ли формула $\mathcal{A}$ тождественно истинной. Очевидно, что данную задачу можно решить алгоритмом Британского музея, иначе говоря, полным перебором. Рассмотрим другую задачу: определить, является ли формула **логики предикатов** тождественно истинной. Для данной задачи алгоритм перебора в общем случае уже не применим, так как множество значений переменных не обязано быть конечным. Но оказывается, для некоторых языков логики предикатов существуют алгоритмы решающие эту задачу. Одним из таких алгоритмов и является алгоритм Тарского, исследованию и реализации которого посвящена данная работа.

Цели работы: изучить, исследовать и описать алгоритм Тарского, реализовать его в виде компьютерной программы, исследовать возможности применения алгоритма Тарского или компонентов его реализации в других задачах.

Задачи:

- Ввести определения, сформулировать и доказать утверждения необходимые для описания алгоритма Тарского и, соответственно, дать описание алгоритма;
- Реализовать компьютерную программу, которая по формуле элементарной алгебры введенной с клавиатуры, строит эквивалентную бесквантворную формулу;
- Проанализировать возможность применения алгоритма Тарского и повторное использование компонентов программы для решения других задач.

В первой части данной работы будет определен язык элементарной алгебры, дано определение элиминации кванторов и сформулировано утверждения о ней. Далее пойдет речь об идеях, на которых основан алгоритм, будут определены таблицы Тарского. Затем будут даны определения полунасыщенной и насыщенной систем многочленов, после чего будет описан метод построения таблиц Тарского, что завершит описание алгоритма Тарского.

Во второй части подробно рассматривается программа, написанная и отлаженная автором работы, а именно описано как происходит распознавание формулы, какие при этом используются алгоритмы, как организованно представление формул, описываются реализации насыщения системы многочленов и построения таблицы Тарского.

1 Алгоритм Тарского

Прежде всего определим область математики, истинность утверждений которой должен проверять алгоритм. Затем зададим язык, на котором записываются эти утверждения. И, наконец, опишем алгоритм, который по формуле описанного языка строит эквивалентную бескванторную формулу.

1.1 Элементарная алгебра

Под элементарной алгеброй понимается та часть общей теории действительных чисел, в которой используются переменные, представляющие собой действительные числа, и константы для всех рациональных чисел, также в которой заданы арифметические операции, такие как «сложение» и «умножение», и отношения сравнения действительных чисел — «меньше», «больше» и «равно». То есть рассматриваются системы алгебраических уравнений и неравенств.

Заметим, что используя декартову систему координат, большую часть всех задач геометрии можно сформулировать как задачи элементарной алгебры.

Например, теорема о пересечении высот треугольника, которая утверждает, что три высоты невырожденного треугольника пересекаются в одной точке, равносильна утверждению: для любых трех точек $A(x_1, y_1)$, $B(x_2, y_2)$ и $C(x_3, y_3)$, не лежащих на одной прямой, существует точка $D(x_4, y_4)$ такая, что $\overrightarrow{AD} \perp \overrightarrow{BC}$, $\overrightarrow{BD} \perp \overrightarrow{AC}$ и $\overrightarrow{CD} \perp \overrightarrow{AB}$.

Или иначе говоря, если $\overrightarrow{AB} \wedge \overrightarrow{AC} \neq 0$, то система

$$\begin{cases} \left(\overrightarrow{AD}, \overrightarrow{BC}\right) = 0 \\ \left(\overrightarrow{BD}, \overrightarrow{AC}\right) = 0 \\ \left(\overrightarrow{CD}, \overrightarrow{AB}\right) = 0 \end{cases}$$

имеет решение относительно переменных x_4, y_4, где $* \wedge *$ — псевдоскалярное произведение векторов, $(*, *)$ — скалярное произведение векторов.

Продолжая эти рассуждения, можно определить формальный язык элементарной алгебры.

1.2 Язык элементарной алгебры

Язык элементарной алгебры — это язык логики первого порядка с сигнатурой

$$\tau = \langle\, \mathbb{Q},\, F,\, P,\, \theta,\, \phi \,\rangle,$$

где

- $\mathbb{Q}$ — множество рациональных чисел, которое является множеством индивидных констант;
- $F = \{+, \cdot\}$ — множество функциональных символов;
- $P = \{<, >, =\}$ — множество предикатных символов;
- $\theta : F \to \mathbb{N}$ такое, что $\theta(+) = 2$ и $\theta(\cdot) = 2$;
- $\phi : P \to \mathbb{N}$ такое, что $\phi(<) = 2$, $\phi(>) = 2$ и $\phi(=) = 2$.

Из определения отображений θ и ϕ видно, что все $f \in F$ являются двухместными функциональными символами, а все $p \in P$ являются двухместными предикатными символами.

Основное множество интерпретации языка L_τ совпадает с множеством действительных чисел $\mathbb{R}$.

Отображение множества индивидных констант в основное множество определяется естественным образом, так как $\mathbb{Q} \subset \mathbb{R}$.Функциональные символы $+$ и $\cdot$ отображаются в сложение и умножение в поле $\mathbb{R}$ соответственно. И предикатные символы $<$, $>$ и $=$ отображаются естественным образом в операции сравнения в $\mathbb{R}$.

Заметим, что множество констант ограничено рациональными числами лишь потому, что компьютер может быстро работать с ними без потери точности, что нельзя сказать про действительные числа.

Таким образом, теорему о пересечении высот на языке элементарной алгебры можно записать следующей формулой:

$$\begin{gathered}(\forall x_1)(\forall y_1)(\forall x_2)(\forall y_2)(\forall x_3)(\forall y_3)\\ ((\neg((x_2 - x_1)\cdot(y_3 - y_1) - (y_2 - y_1)\cdot(x_3 - x_1) = 0)) \rightarrow\\ (\exists x_4)(\exists y_4)(((x_4 - x_3)\cdot(x_2 - x_1) + (y_4 - y_3)\cdot(y_2 - y_1) = 0)\&\\ ((x_4 - x_2)\cdot(x_1 - x_3) + (y_4 - y_2)\cdot(y_1 - y_3) = 0)\&\\ ((x_4 - x_1)\cdot(x_3 - x_2) + (y_4 - y_1)\cdot(y_3 - y_2) = 0))).\end{gathered}$$

Также отметим, что нет необходимости вводить в языке такие операции как вычитание, деление и возведение в степень, так как используя свойства поля и операций сравнения их можно выразить через сложение и умножение:

$$a - b = a + (-1)\cdot b; \quad \frac{a}{b} > 0 \Leftrightarrow (a > 0 \,\&\, b > 0) \vee (a < 0 \,\&\, b < 0); \quad x^2 = x \cdot x.$$

Получившаяся формула содержит кванторы. Если мы могли бы «сократить» кванторы с соответствующими переменными, то получили бы формулу логики высказываний. То есть задачу определения истинности формулы языка элементарной алгебры могли бы свести к задаче определения истинности формулы логики высказываний, которая является алгоритмически разрешимой.

1.3 Элиминация кванторов

Процесс «сокращения» кванторов принято называть элиминацией кванторов.

Определение 1.1. Элиминация кванторов — это процесс, порождающий по заданной логической формуле, другую, эквивалентную ей бескванторную формулу, то есть свободную от вхождений кванторов.

Пусть алгоритм A такой, что $A\left((Qx)\,\mathcal{A}\right) = \mathcal{B}$, где $\mathcal{A}$ и $\mathcal{B}$ — бескванторные формулы языка элементарной алгебры, и формулы $(Qx)\mathcal{A}$ и $\mathcal{B}$ эквивалентны, а Q — квантор. Тогда верно следующее утверждение:

Утверждение 1.1. Если алгоритм A существует, то существует алгоритм B такой, что для любой формулы $\mathcal{A}$ языка элементарной алгебры $B\left(\mathcal{A}\right)$ — бескванторная формула, эквивалентная $\mathcal{A}$.

Доказательство. Определим алгоритм B следующим образом:

- Если $\mathcal{A}$ — бескванторная формула, то $B\left(\mathcal{A}\right) = \mathcal{A}$;

- Если $\mathcal{A} = (Qx)\,\mathcal{B}$, то $B(\mathcal{A}) = A((Qx)\,B(\mathcal{B}))$.

  Формула $B(\mathcal{B})$ — бескванторная по построению B, следовательно алгоритм A можно применить к формуле $((Qx)\,B(\mathcal{B}))$.

  Формула $B(\mathcal{B})$ эквивалентна $\mathcal{B}$, следовательно, формула $(Qx)\,\mathcal{B}$ эквивалентна $(Qx)\,B(\mathcal{B})$, а значит $\mathcal{A}$ эквивалентна $B(\mathcal{A})$.

  Также заметим, что длина формулы $\mathcal{B}$ строго меньше длины формулы $\mathcal{A}$.

- Если $\mathcal{A}$ не удовлетворяет предыдущим условиям, то

  – либо $\mathcal{A} = \neg\mathcal{B}$, тогда $B(\mathcal{A}) = \neg B(\mathcal{B})$,

  – либо $\mathcal{A} = \mathcal{B} * \mathcal{C}$, тогда $B(\mathcal{A}) = B(\mathcal{B}) * B(\mathcal{C})$, где $* \in \{\vee, \&, \to\}$.

  При этом длины формул $\mathcal{B}$ и $\mathcal{C}$ меньше длины формулы $\mathcal{A}$.

Алгоритм B определен рекурсивно, при этом на каждом этапе на вход B подаётся формула меньшей длины, следовательно, алгоритм B является конечным, а на каждом шаге выход алгоритма — бескванторная эквивалентная формула. ◀

Отметим, что доказательство не использует информацию о сигнатуре языка, следовательно данное утверждение верно и для других языков логики предикатов.

Таким образом, чтобы построить алгоритм элиминации кванторов, достаточно построить алгоритм A. Для языка элементарной алгебры таким алгоритмом является алгоритм Тарского.

1.4 Алгоритм Тарского

Прежде чем перейти к рассмотрению алгоритма, необходимо сделать ряд замечаний и предложений.

Термы в языке элементарной алгебры — это многочлены с рациональными коэффициентами от действительных переменных Тогда очевидно, что выражения

$$f < g, \quad f = g, \quad f > g$$

равносильны выражениям

$$f - g < 0, \quad f - g = 0, \quad f - g > 0$$

соответственно, где f и g — термы. Поэтому, не нарушая общности рассуждений, можно считать, что все атомарные формулы имеют вид:

$$f < 0, \quad f = 0, \quad f > 0.$$

Поэтому мы будем говорить, что нас интересует только знак многочлена: больше, меньше или равен нулю.

Очевидно, что нулевой многочлен и многочлены нулевой степени не меняют свой знак на всей области определения, их знак определяется тривиальным образом.

Известно, что многочлены от одной переменной задают непрерывные функции, следовательно, эти функции сохраняют свой знак на интервалах между корнями. Значит, чтобы уметь определять знак значения многочлена в произвольной точке, достаточно знать знак многочлена:

- в его корнях, значение в которых, очевидно, равно нулю;

- в любой точке каждого из интервалов, на которые разбивается область определения корнями.

Во-первых, по свойствам многочленов, множество корней ненулевого многочлена конечно. Во-вторых, конечное число точек разбивают числовую прямую на конечное число интервалов. Таким образом, необходимо знать знак многочлена лишь в **конечном** наборе точек.

Рассмотрим формулу $\mathcal{A} = (Qx)(f(x)\,\rho\,0)$, где Q — квантор, $f(x)$ — многочлен от одной переменной, ρ — предикат. Из наших рассуждений следует, что формуле $\mathcal{A}$ эквивалентна следующая **бескванторная** формула:

$$\mathcal{B} = \begin{cases} \bigvee\limits_{x_0 \in X} (f(x_0)\,\rho\,0), & \text{если } Q = \exists \\ \bigwedge\limits_{x_0 \in X} (f(x_0)\,\rho\,0), & \text{если } Q = \forall \end{cases}$$

где X — конечное множество точек.

Предположим, что мы умеем находить все корни многочлена.

По теореме Ролля о нуле производной, для любой пары корней x_1, x_2 вещественной, непрерывной и дифференцируемой функции существует такая точка ξ, лежащая между x_1, x_2, что производная функции в точке ξ обращается в ноль. Производная многочлена есть многочлен, тогда, исходя из нашего предположения, мы можем вычислить корни производной многочлена. Поэтому в качестве точек на интервале между корнями будем использовать корни производной этого многочлена.

Для интервала справа от всех корней и для интервала слева от всех корней предлагается взять любую точку, которая соответственно больше или меньше всех корней многочлена.

Заметим, что множество корней многочлена

$$\prod_{i=1}^{n} f_i(x)$$

совпадает с объединением множеств корней многочленов $f_1(x), ..., f_n(x)$. Поэтому в качестве точек между корнями этого многочлена можно рассматривать корни многочлена

$$f_0(x) = \left(\prod_{i=1}^{n} f_i(x) \right)'.$$

Теперь нетрудно перейти от атомарных формул, к формулам общего вида. Пусть $\mathcal{A} = (Qx)(\Phi(x))$, где $\Phi(x)$ — бескванторная формула, которая может содержать вхождения лишь переменной x. Тогда формула

$$\mathcal{B} = \begin{cases} \bigvee\limits_{x_0 \in X} \Phi(x_0), & \text{если } Q = \exists \\ \bigwedge\limits_{x_0 \in X} \Phi(x_0), & \text{если } Q = \forall \end{cases}$$

свободна от вхождений кванторов и эквивалента формуле $\mathcal{A}$, где X — объединение множества всех корней произведения всех многочленов, входящих в $\Phi(x)$, и множества всех корней производной этого произведения.

Для того, чтобы отказаться от предположения, что мы умеем находить корни произвольного многочлена, введем ряд понятий, в том числе понятие таблица Тарского.

1.4.1 Таблица Тарского

Упорядочим выбранные точки $X = \{x_1, ..., x_s\}$ по возрастанию и запишем значения многочленов в этих точках в таблицу:

	x_1	...	x_j	...	x_s
f_1	$f_1(x_1)$	...	$f_1(x_j)$	...	$f_1(x_s)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
f_i	$f_i(x_1)$	...	$f_i(x_j)$	...	$f_i(x_s)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
f_n	$f_n(x_1)$	...	$f_n(x_j)$	...	$f_n(x_s)$

Как отмечалось ранее, нас интересуют только знаки многочленов в этих точках, поэтому в ячейках таблицы оставим лишь символ знака значения:

	x_1	...	x_j	...	x_s
f_1	ε_{11}	...	ε_{1j}	...	ε_{1s}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
f_i	ε_{i1}	...	ε_{ij}	...	ε_{is}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
f_n	ε_{n1}	...	ε_{nj}	...	ε_{ns}

где

$$\varepsilon_{ij} = \begin{cases} +, & \text{если } f_i(x_j) > 0 \\ 0, & \text{если } f_i(x_j) = 0 \\ -, & \text{если } f_i(x_j) < 0 \end{cases}.$$

Таблицы такого вида будем называть **таблицами Тарского**.

Утверждение 1.2. Знаки + и − не могут стоять в двух соседних по горизонтали клетках таблицы Тарского.

Доказательство. Предположим противное. Пусть многочлен f_i принимает в точке x_j положительное значение, а в точке x_{j+1} отрицательное. Многочлены задают непрерывные функции, тогда, по теореме Коши о нулях непрерывной функции, существует такая точка $\xi \in (x_j, x_{j+1})$, что $f(\xi) = 0$, то есть ξ — корень многочлена. Но тогда $\xi \in X$ по построению множества X, при этом значения x_j упорядочены. Получили противоречие. ◀

Утверждение 1.3. [1] Если многочлен отличен от тождественного нуля, то в строке таблицы Тарского, соответствующей этому многочлену, в соседних по горизонтали клетках не могут стоять два символа 0.

Доказательство. Предположим противное. Эти точки являются корнями многочлена, которым отмечена строка. Тогда множество X должно содержать точку между этими корнями, что противоречит тому, что корни являются соседними точками в таблице Тарского. ◀

Имея таблицу Тарского, нетрудно вычислить истинностное значение формулы $\Phi(x_i)$, так как для определения истинностного значения атомарной формулы достаточно посмотреть на соответствующую клетку таблицы. При этом уже нет необходимости знать точки $x_1, ..., x_n$. А зная истинностное значение формулы, можно построить эквивалентную бескванторную:

- если формула истинна — то можно использовать любую тождественно истинную формулу, например, $0 = 0$;
- если формула ложна — то можно использовать любую тождественно ложную формулу, например $0 = 1$.

До сих пор не обсуждалось, как искать корни многочленов. Оказывается, таблицу Тарского можно построить не находя ни одного корня, если рассмотреть системы многочленов особого вида.

1.4.2 Насыщенная система

Определение 1.2. [1] Система функций называется **полунасыщенной**, если вместе с каждой функцией, отличной от постоянной функции, она содержит и ее производную.

Утверждение 1.4. [1] Каждую конечную систему многочленов можно расширить до конечной полунасыщенной системы.

Доказательство. Поместим все многочлены в очередь. Далее, пока очередь не пуста, извлекаем многочлен из очереди. Этот многочлен добавляется в множество-ответ. Если степень многочлена больше единицы, то его производная помещается в очередь.

Заметим, что на каждом шаге суммарная степень многочленов в очереди строго убывает, поэтому будет выполнено конечное число итераций алгоритма и в множестве-ответ будет конечное число многочленов. ◀

Определение 1.3. [1] Полунасыщенная система многочленов $p_1(x), ..., p_n(x)$ называется **насыщенной**, если вместе с каждыми двумя многочленами $p_i(x)$ и $p_j(x)$ такими, что $0 < deg(p_j(x)) \leq deg(p_i(x))$, она содержит и остаток $r(x)$ от деления $p_i(x)$ на $p_j(x)$.

Утверждение 1.5. [1] Каждую конечную систему многочленов можно расширить до конечной насыщенной системы.

Доказательство. Поместим все многочлены в очередь. Далее, пока очередь не пуста, извлекаем многочлен f из очереди. Этот многочлен добавляется в множество-ответ. Если степень многочлена больше единицы, то его производная помещается в очередь. Для каждого многочлена g из множества-ответ степени больше 0 помещаем в очередь остатки от деления f на g и g на f.

Если на каждом шаге извлекать из очереди многочлен наибольшей степени, то количество многочленов наибольшей степени будет уменьшаться, а вместе с ним будет уменьшаться наибольшая степень многочленов, так как степень остатка от деления меньше степени обоих многочленов. Таким образом, будет выполнено лишь конечное число итераций алгоритма и в множестве-ответ будет конечное число многочленов. ◀

Утверждение 1.6. [1] Если $p_1(x), ..., p_{n-1}(x), p_n(x)$ — насыщенная система многочленов, и

$$deg(p_1(x)) \leq ... \leq deg(p_{n-1}(x)) \leq deg(p_n(x)),$$

то система $p_1(x), ..., p_{n-1}(x)$ также является насыщенной.

Доказательство. Так как степень производной многочлена и степень остатка от деления меньше степени самого многочлена, то $p_n(x)$ не может являться ни производной, ни остатком от деления, поэтому после его удаления система не перестанет быть насыщенной. ◀

Утверждение 1.7. [1] Если $p_1(x), ..., p_{n-1}(x), p_n(x)$ — насыщенная система многочленов, и

$$deg(p_1(x)) \leq ... \leq deg(p_{n-1}(x)) \leq deg(p_n(x)),$$

то для любого натурального $m < n$ система $p_1(x), ..., p_{n-m}(x)$ также является насыщенной.

Доказательство. Необходимо m раз применить утверждение 1.6. ◀

Утверждение 1.8. Если $p_1(x), ..., p_n(x)$ — насыщенная система многочленов, и

$$deg(p_1(x)) \leq deg(p_i(x)), \text{ где } i = 2, 3, ..., n,$$

то $deg(p_1(x)) < 1$.

Доказательство. Если предположить противное, то с одной стороны, система должна содержать многочлен $p_1'(x)$, степень которого меньше $deg(p_1(x))$, а с другой стороны, степени всех многочленов должны быть не меньше $deg(p_1(x))$, противоречие. ◀

1.4.3 Метод построения таблицы Тарского

Пусть $p_1(x), ..., p_{n-1}(x), p_n(x)$ — насыщенная система многочленов, и многочлены упорядочены по не убыванию степени.

Рассмотрим подсистему из одного элемента $p_1(x)$. Согласно утверждению 1.7, система $p_1(x)$ является насыщенной, тогда многочлен $p_1(x)$ представляет собой константу, так как его степень меньше единицы, согласно утверждению 1.8. В таком случае знак многочлена в любой точке совпадает со знаком этой константы. Таблица Тарского для одного многочлена имеет вид:

	$-\infty$	$+\infty$
p_1	ε	ε

Символами $-\infty$ и $+\infty$ обозначены точки, которые заведомо расположены левее и правее всех корней соответственно. Выбирать конкретные значения для этих точек не нужно, вместо этого предлагается считать знак предела многочлена при x стремящемся к $-\infty$ и $+\infty$. Поэтому в точке $+\infty$ знак многочлена совпадает со знаком старшего коэффициента, а в точке $-\infty$ знак зависит от четности степени многочлена:

- если четная, то совпадает со знаком старшего коэффициента;
- иначе равен знаку противоположному к знаку старшего коэффициента.

В таблице всего два столбца, поэтому верно утверждение: для каждого столбца j, за исключением самого правого и самого левого, в этой таблице существует ненулевой многочлен $p_i(x)$ такой, что $\varepsilon_{i,j} = 0$.

Индуктивное предположение: пусть для насыщенной системы $p_1(x), ..., p_{k-1}(x)$ уже построена таблица Тарского:

	$-\infty$		...		...		$+\infty$
p_1	$\varepsilon_{1,1}$	$\varepsilon_{1,2}$	...	$\varepsilon_{1,j}$	...	$\varepsilon_{1,s-1}$	$\varepsilon_{1,s}$
p_2	$\varepsilon_{2,1}$	$\varepsilon_{2,2}$	...	$\varepsilon_{2,j}$	...	$\varepsilon_{2,s-1}$	$\varepsilon_{2,s}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
p_{k-1}	$\varepsilon_{k-1,1}$	$\varepsilon_{k-1,2}$	...	$\varepsilon_{k-1,j}$	...	$\varepsilon_{k-1,s-1}$	$\varepsilon_{k-1,s}$

И для каждого столбца j, за исключением самого правого и самого левого, в этой таблице существует ненулевой многочлен $p_i(x)$ такой, что $\varepsilon_{i,j} = 0$.

К этой таблице добавим строку для многочлена $p_k(x)$, записав знаки для крайних столбцов.

	$-\infty$		...		...		$+\infty$
p_1	$\varepsilon_{1,1}$	$\varepsilon_{1,2}$	...	$\varepsilon_{1,j}$	...	$\varepsilon_{1,s-1}$	$\varepsilon_{1,s}$
p_2	$\varepsilon_{2,1}$	$\varepsilon_{2,2}$	...	$\varepsilon_{2,j}$	...	$\varepsilon_{2,s-1}$	$\varepsilon_{2,s}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
p_{k-1}	$\varepsilon_{k-1,1}$	$\varepsilon_{k-1,2}$	...	$\varepsilon_{k-1,j}$	...	$\varepsilon_{k-1,s-1}$	$\varepsilon_{k-1,s}$
p_k	$\varepsilon_{k,1}$	?	...	?	...	?	$\varepsilon_{k,s}$

Для каждого столбца j рассмотрим многочлен $p_i(x)$ такой, что $\varepsilon_{i,j} = 0$. Этот многочлен существует и отличен от тождественного нуля в силу индуктивного предположения.

Утверждение 1.9. Пусть $f(x)$ и $g(x)$ — ненулевые многочлены. Если $g(a) = 0$, то $f(a) = r(a)$, где $r(x)$ — остаток от деления многочлена $f(x)$ на $g(x)$.

Доказательство. Многочлен $r(x)$ — остаток от деления, тогда $f(x) = q(x)g(x)+r(x)$, подставив a получим $f(a) = q(a)g(a) + r(a) = q(a) \cdot 0 + r(a) = r(a)$. ◀

Найдём $p_t(x)$ — остаток от деления $p_k(x)$ на $p_i(x)$. Система многочленов насыщена, поэтому многочлен $p_t(x)$ уже добавлен в таблицу, тогда, применив утверждению 1.9, можем вычислить $\varepsilon_{k,j} = \varepsilon_{t,j}$. Таким образом, заполняется вся нижняя строка.

Просмотрим значения в нижней строке. Может случиться так, что в соседних клетках стоят знаки $+$ и $-$.

$+$	$-$

$-$	$+$

В таком случае необходимо добавить новый столбец между теми столбцами, в которых находятся эти клетки. Понятно, что в новом столбце нижняя клетка заполняется нулем, так как этот новый столбец заполняется для корня, существования которого следует из теоремы Коши о нулях непрерывной функции.

Рассмотрим как заполнять новый столбец для остальных строк.

$+$	?	$+$

$+$	?	0

$+$	$+$	$+$

$+$	$+$	0

Тогда вместо символа ? ставится знак $+$, так как непрерывная функция сохраняет свой знак на интервалах между корнями.

0	?	$+$

0	?	$-$

0	$+$	$+$

0	$-$	$-$

В этих случаях ставится знак $+$ или $-$ соответственно.

$-$	?	0

$-$	?	$-$

$-$	$-$	0

$-$	$-$	$-$

В этих случаях знак $-$. И наконец, в случае

0	?	0

0	0	0

ставится 0, так как эта строка точно соответствует нулевому многочлену.

А случаи

$+$	?	$-$

$-$	?	$+$

невозможны по построению таблицы Тарского.

Таким образом, удалось построить таблицу Тарского для насыщенной системы многочленов $p_1(x), ..., p_{k-1}(x), p_k(x)$, при этом для каждого столбца найдется многочлен, на пересечении строки которого с выбранным столбцом в клетке записан символ 0.

1.4.4 Алгоритм для формулы вида $(Qx)\Phi(x)$

Все готово, чтобы описать алгоритм Тарского для формулы $\mathcal{A} = (Qx)\Phi(x)$:

1. Составить список всех многочленов $f_1(x), ..., f_n(x)$, входящих в $\Phi(x)$ и отличных от тождественного нуля;
2. Добавить к этому списку многочлен
$$f_0(x) = \left(\prod_{i=1}^{n} f_i(x)\right)';$$
3. Расширить этот список до насыщенной системы $p_1(x), ..., p_m(x)$, упорядоченной по не убыванию степени;
4. Построить таблицу Тарского;
5. Вычислить истинностное значение $\Phi(x)$ для каждого из столбцов таблицы;
6. Если $Q = \exists$, то формула $\mathcal{A}$ истинна тогда и только тогда, когда хотя бы одно из вычисленных значений истинно.

   Если $Q = \forall$, то формула $\mathcal{A}$ истинна тогда и только тогда, когда все вычисленные значения истинны.

1.4.5 Алгоритм для формулы вида $(Qx)\Phi(x, a_1, ..., a_l)$

Оказывается, в случае, когда формула имеет вид $(Qx)\Phi(x, a_1, ..., a_l)$, нужно лишь немного модифицировать алгоритм. Во-первых, коэффициенты многочленов теперь не из $\mathbb{Q}$, а из поля частных целостного кольца $\mathbb{Q}[a_1, ..., a_l]$. Во-вторых, нельзя говорить о знаках таких коэффициентов, поэтому каждый раз, когда необходимо определить знак коэффициента, придется разбирать три случая: коэффициент меньше нуля, больше нуля или равен нулю. Поэтому будет построено дерево разбора случаев. В листьях этого дерева все знаки определены и можно построить таблицу Тарского. Если по таблице получается, что формула истинна, тогда все предположения, сделанные в ходе разбора случаев, выписываются в виде конъюнкции. Результатом же работы алгоритма будет дизъюнкция всех таких конъюнкций для каждого пути в дереве разбора.

1.5 Пример работы алгоритма

Рассмотрим формулу $(\forall x)(y < 0 \rightarrow x^2 > y)$ и построим эквивалентную ей бесквантерную формулу с помощью алгоритма Тарского. Многочлены y и $x^2 - y$ входят в данную формулу. Далее нужно выяснить все ли многочлены отличны от тождественного нуля, поэтому рассмотрим два случая:

1. $y = 0$, тогда из списка исключается многочлен $y \equiv 0$ и добавляется многочлен $2x$, при этом $x^2 - y \equiv x^2$;

2. $y < 0$ или $y > 0$, тогда в систему добавляется многочлен $2yx$.

Переходим к насыщению системы:

1. система $2x, x^2$ дополняется до $0, 2, 2x, x^2$;

2. система $y, 2yx, x^2 - y$ дополняется до $0, y, -y, 2y, 2, 2yx, 2x, x^2 - y$.

Построим таблицы Тарского:

1. $y = 0$, система $0, 2, 2x, x^2$:

	$-\infty$	$+\infty$
0	0	0
2	+	+

	$-\infty$		$+\infty$
0	0	0	0
2	+	+	+
$2x$	−	0	+

	$-\infty$		$+\infty$
0	0	0	0
2	+	+	+
$2x$	−	0	+
x^2	+	0	+

2. $y < 0$, система $0, y, -y, 2y, 2, 2yx, 2x, x^2 - y$:

	$-\infty$	$+\infty$
0	0	0
y	−	−
$-y$	+	+
$2y$	−	−
2	+	+

	$-\infty$		$+\infty$
0	0	0	0
y	−	−	−
$-y$	+	+	+
$2y$	−	−	−
2	+	+	+
$2yx$	+	0	−
$2x$	−	0	+
$x^2 - y$	+	+	+

3. $y > 0$, система $0, y, -y, 2y, 2, 2yx, 2x, x^2 - y$:

	$-\infty$	$+\infty$
0	0	0
y	+	+
$-y$	−	−
$2y$	+	+
2	+	+

	$-\infty$		$+\infty$
0	0	0	0
y	+	+	+
$-y$	−	−	−
$2y$	+	+	+
2	+	+	+
$2yx$	−	0	+
$2x$	−	0	+

	$-\infty$				$+\infty$
0	0	0	0	0	0
y	+	+	+	+	+
$-y$	−	−	−	−	−
$2y$	+	+	+	+	+
2	+	+	+	+	+
$2yx$	−	−	0	+	+
$2x$	−	−	0	+	+
$x^2 - y$	+	0	−	0	+

Нетрудно убедиться в том, что для каждого случая, для каждого столбца формула $(y < 0 \rightarrow x^2 > y)$ истинна. В результате, на выходе алгоритма получим формулу

$$(y = 0 \vee y < 0 \vee y > 0).$$

1.6 Теорема Тарского

Таким образом, нами была доказана следующая теорема.

Теорема 1.1 (Альфред Тарский)**.** Для любой формулы $\mathcal{A}$ языка элементарной алгебры существует эквивалентная ей бесквантворная формула этого же языка.

Алгоритм, предложенный А. Тарским в его работе [7], записывался иначе и был менее понятен — формулы приводились к нормальным формам, явно строились эквивалентные формулы, таблицы не строились. Но и цель была не предложить «хороший» алгоритм, а доказать, что элементарная алгебра допускает элиминацию кванторов. В последующие годы велась работа по упрощению и усовершенствованию алгоритма, особенно в случае вхождений свободных переменных, и в результате этой работы алгоритм приобрел такой вид. Современное описание алгоритма доступнее для понимания, что упрощает его программную реализацию.

2 Программная реализация

После изучения алгоритма Тарского была начата работа по реализации этого алгоритма, но не для произвольных формул, а для формул без параметров.

Определение 2.1. Формула $\mathcal{A}$ называется **формулой без параметров**, если для любой ее подформулы вида $(Qx)\mathcal{B}$, формула $\mathcal{B}$ свободна от вхождений кванторов и от переменных, возможно, за исключением переменной x.

Для этого были выбраны язык C#, платформа .NET Core 3.1 и спецификация .NET Standard 2.1 [6]. Такой выбор обусловлен рядом причин:

- Язык C# — это объектно-ориентированный язык программирования, а данная парадигма программирования позволяет абстрактно описывать объекты, в том числе и математические объекты;
- .NET Core и .NET Standard — это современные, развивающиеся и востребованные кроссплатформенные технологии с открытым исходным кодом;
- Личные предпочтения автора.

Для поэтапного создания программы, были сформулированы и решены следующие задачи:

- Разработать библиотеку классов для объектов языка элементарной алгебры: символы алфавита, термы, формулы;
- Реализовать ввод логических формул, то есть решить задачу лексического анализа, решить задачу синтаксического анализа;
- Разработать библиотеку классов для рациональных чисел и многочленов от одной вещественной переменной с рациональными коэффициентами;
- Реализовать алгоритм Тарского для формул от одной переменной;
- Реализовать алгоритм элиминации кванторов (утверждение 1.1) для формул без параметров;
- Провести тестирование модулей, интеграционное и end-to-end тестирования;
- Собрать все библиотеки и модули в единый программный комплекс.

Написание программы осуществлялось в среде разработки Microsoft Visual Studio 2019 Community с установленным дополнением ReSharper. Обе программы доступны по специальным студенческим лицензиям для использования в некоммерческих целях.

2.1 Представление формул

Необходимо было начать с того, что создать абстракции для всех объектов языка элементарной алгебры: символы алфавита, термы, формулы. Для этого была написана библиотека классов IOLib, в которой определены следующие пространства имён:

- в IOLib.Exceptions расположены классы для ошибок, исключительных ситуаций, возникающих при разборе входной строки;

- в IOLib.Input — классы и интерфейсы, определяющие токенизацию, парсинг и трансляцию;
- в IOLib.Language — определены классы, моделирующие объекты языка, а именно символы, термы формулы;
- в IOLib.Output — интерфейс и классы для обратного преобразования формулы в строку в LaTeX-like формате.

2.1.1 Класс Symbol

Начнём с класса Symbol. Он предоставляет представление в виде строки (string), реализует интерфейс IEquatable<Symbol>, чтобы иметь возможность сравнивать символы. Сравнение происходит путём сравнения строковых представлений символов.

Далее следует рассмотреть статический класс Alphabet, который предоставляет множество полей, каждое из котоых имеет тип Symbol и моделирует один из символов входного языка. Также класс предоставляет несколько методов для работы с буквами и цифрами, а именно получение соответствующего символа по объекту типа char, проверка является ли данный объект типа Symbol буквой или цифрой.

Было принято решение отказаться от архитектуры, используемой в предыдущих работах, когда под каждый символ создавался класс. Это приводило к тому, что код содержал много однотипных сущностей без методов, полей и свойств. Новая архитектура решает задачи, которые обычно решают с помощью паттерна одиночка, который в свою очередь был не применим в предыдущем подходе из-за наследования.

2.1.2 Класс Term

Прежде чем перейти к описанию формул, необходимо описать класс для термов языка элементарной алгебры. При определении терма выделяется три случая: константа, переменная, функция от термов. Поэтому абстрактный класс Term является базовым для трех классов. Чтобы для формул находить список переменных, входящих в терм, объявлено абстрактное свойство ObjectVariables, которое очевидным образом реализуют наследники.

Первый и второй классы оборачивают константы и переменные, а третий хранит функцию и массив аргументов, то есть другие термы.

2.1.3 Класс Formula

И наконец, объект, для которого были написаны все ранее перечисленные классы – формула языка элементарной алгебры. Вновь, исходя из определения формулы получается такая иерархия классов: во главе абстрактный класс Formula и три класса-наследника.

Класс PredicateFormula описывает формулы вида $p\,(t_1, ..., t_n)$, где p – n-местный предикат, а t_i — терм. Класс PropositionalConnectiveFormula, в котором объявлены поля для пропозициональной связки и аргументов, описывает формулы вида $\mathcal{A} * \mathcal{B}$, где $* \in \{\&, \vee, \rightarrow\}$, или вида $\neg\mathcal{A}$. И для формул вида $(Qx)(\mathcal{A})$ реализован класс QuantifierFormula.

В результате, описаны все объекты языка элементарной алгебры. При этом данная система допускает расширение, например, можно добавлять логические связки, функции или вводить новые предикаты.

2.2 Система ввода

Далее необходимо реализовать методы, преобразующие введенную пользователем строку в формулу, то есть в экземпляр класса Formula, или, сообщают о том, что данная строка не является формулой. Аналогичная задача, но для формул исчисления высказываний, была успешно решена в предыдущей работе [3]. Однако и в этом аспекте было решено изменить подход к решению задачи. Отказаться от алгоритма Дейкстры и ОПЗ в пользу классического метода решения задачи синтаксического анализа — разбор формулы по грамматике нисходящим разбором.

Задачу лексического анализа можно решить с помощью конечных автоматов. Интерфейс ITokenizer определяет функцию, которая будет решать эту задачу, то есть по строке на входы выдавать последовательность символов на выходе.

Была задана входная грамматика, каждый символ закодирован строкой также, как это делается в системе LaTeX и получен набор лексем или ещё их называют токенами. Далее был придуман и реализован следующий алгоритм: по токенам строится префиксное дерево, затем это дерево преобразуется в автомат Мили, то есть в автомат с выходом.

Синтаксический анализ осуществляется алгоритмом нисходящего разбора. В общем случае этот алгоритм является экспоненциальным, однако для данной грамматики можно показать, что алгоритмы понадобиться шагов не более чем константа умноженная на квадрат от количества символ в строке.

2.3 Реализация алгоритма Тарского

Переходим непосредственно к реализации алгоритма построения бескванторной формулы, а точнее к рассмотрению библиотеки SimpleTarskiAlgorithmLib.

Первым делом рассмотрим перечисление Sign, так как именно знаки чисел и многочленов являются важной и при этом самой простой частью алгоритма. В этом перечислении определены следующие значения:

- NotNumber — объект не имеет знака;
- LessZero — меньше нуля;
- MoreZero — больше нуля;
- Zero — нуль;
- NotMoreZero — не больше нуля;
- NotLessZero — не меньше нуля;
- NotZero — не нуль;
- Undefined — знак может быть любым.

Так как рассматриваются многочлены с коэффициентами из $\mathbb{Q}$, необходимо реализовать структуру для рациональных чисел. Для этого достаточно в качестве целых чисел взять структуру BigInteger и вспомнить как определяются рациональные числа и операции над ними в алгебре. Поэтому для этой структуры определены поля для числителя и знаменателя, а также поле для знака числа. Естественным образом реализованы арифметические операции сложение, вычитание, умножение, деление и возведение в натуральную степень.

Для реализации класса для многочленов вновь необходимо вспомнить как они определяются в алгебре и реализовать эти определения в виде программы. Поэтому в классе Polynomial определен массив из рациональных чисел, переменная типа VariableName и поля для знака многочлена. Как в алгебре старший коэффициент многочлена отличен от нуля, так и элемент массива с наибольшим индексом не должен быть равным нулю. Для многочленов определены операции сложения, вычитания, умножения, деление с остатком, формальное дифференцирование и возведение в натуральную степень.

Переходим к задаче насыщения системы многочленов. Для её решения создан статический класс SimpleSaturator, а точнее его метод Saturate. На вход ему поступает последовательность многочленов. Затем он из них выбирает все ненулевые, вычисляет производную их произведения и помещает выбранные многочлены и производную в очередь. Далее пока очередь не пуста происходит следующее:

- Извлекаем из очереди первый многочлен P. Если он уже добавлен в результирующее множество, то есть в HashSet<Polynomial>, то переходим к следующей итерации цикла;
- Добавляем в очередь производную этого многочлена;
- Для каждого многочлена Q из результирующего множества добавляем его остаток от деления на P, если $deg(Q) \geq deg(P)$, и остаток от деления P на Q, если $deg(P) \geq deg(Q)$.

После этапа насыщения системы следует этап построения таблицы Тарского, поэтому необходимо реализовать структуру данных соответствующую свойствам таблицы. Во-первых, довольно часто будут добавляться столбцы, поэтому предлагается рассматривать таблицы как связный список столбцов, что позволяет наиболее эффективно добавлять новые столбцы. Во-вторых, строки добавляются только в конец таблицы, поэтому столбцы представляют собой список знаков — List<Sign>, так как добавление в конец списка и обращение к элементу списка происходят достаточно быстро.

Рассмотрим последнюю библиотеку — SimpleTarskiAlgorithmRunner. Именно класс SimpleTarskiAlgorithm, расположенный в этой библиотеке, связывает формулы и алгоритм Тарского. QuantifiersElimination — единственный публичный метод этого класса. Он реализует алгоритм B, описанный в утверждении 1.1, а алгоритм A, то есть алгоритм Тарского для формулы вида $(Qx)\Phi(x)$, реализован в виде метода TarskiEliminate.

На этом заканчивается описание программы. Исходный код проекта размещен в виде репозитория на GitHub, ссылку на который можно найти в приложении А. Еще в репозитории можно найти проекты с юнит-тестами, так как весь код отлаживался, а наиболее сложные его части тестировались.

2.4 Демонстрация работы

Чтобы продемонстрировать работу программы было создано простое консольное приложение. Пользователю предлагается ввести формулу. После того как пользователь ввел формулу ему выводится сообщение о том, в какой файл записан результат работы программы. Далее (рис. 1) приведено несколько примеров содержания таких выходных файлов.

Для того, чтобы реализовать алгоритм Тарского для формул общего вида, необходимо, во-первых, реализовать многочлены с параметрами, во-вторых, насыщение системы. Если с многочленами сложностей не возникает (в проекте уже реализован соответствующий класс), то насыщение системы таких многочленов оказалось нетривиальной задачей, так как требует разбора случаев знака коэффициентов. При этом построение таблицы и остальные части, за редким исключением, не отличаются от того, что уже было реализовано. Поэтому можно сказать, что в данной работе реализован базовый, но наиболее важный и трудоёмкий случай.

```
\exists(x, x^2 = 2025)
Entered formula: (∃x)((x^2)=2025)
Result: TRUE()
```

а)

```
\forall(x, x^2>0 \lor x=0) \land x+y>0
Entered formula: ((∀x)(((x^2)>0)∨(x=0))&((x+y)>0))
Result: ((x+y)>0)
```

б)

```
\exists(x, (x < -1 \lor x > 2) \land 5*x^2 - (11/2) * x - 8 = 0)
Entered formula: (∃x)(((x<-(1))∨(x>2))&((((5*(x^2))-((11/2)*x))-8)=0))
Result: FALSE()
```

в)

Рис. 1: Примеры выходных данных.

Заключение

Таким образом, цель данной работы достигнута, все поставленные задачи решены. В работе описан языка элементарной алгебры, приведены примеры задач элементарной алгебры, достаточно подробно описан алгоритм элиминации кванторов в этом языка — алгоритм Тарского. По мимо этого, алгоритм был реализован в виде компьютерной программы для достаточно важного случая — для формул без параметров. Эта программа включает в себя систему ввода формул языка элементарной алгебры, библиотеки классов для представления объектов этого языка и собственно реализацию алгоритма Тарского.

В заключении хочется отметить, что алгоритм Тарского далеко не самый эффективный алгоритм, но он был первым в своем роде. Именно сконструировав этот алгоритм Альфред Тарский доказал, что элементарная алгебра допускает элиминацию кванторов, хотя до этого многие годы это считалось невозможным.

Для систем компьютерного доказательства, которые в ближайшем будущем станут очень востребованными (формальная верификация компьютерных программ), алгоритм Тарского вряд ли применим из-за крайне высокой трудоемкости. Поэтому можно продолжать работу в данному направлении и изучать другие алгоритмы элиминации кванторов. А с точки зрения математики, интересен вопрос, а какие ещё языки допускают элиминацию кванторов? Поэтому автор продолжит работу в данном направлении.

Список литературы

[1] Алгоритм Тарского. Лекция 1 // Лекториум. URL: https://www.lektorium.tv/lecture/31079 (дата обращения: 01.12.2019).

[2] Алгоритм Тарского. Лекция 2 // Лекториум. URL: https://www.lektorium.tv/lecture/31080 (дата обращения: 01.12.2019).

[3] Гибадулин Р. А. Алгоритм поиска вывода в Исчислении Высказываний и его программная реализация // Современные проблемы математики и информатики : сборник научных трудов молодых ученых, аспирантов и студентов. / Яросл. гос. ун-т им. П. Г. Демидова. — Ярославль: ЯрГУ, 2019. — Вып. 19. — С. 28–37.

[4] Дурнев, В. Г. Элементы теории множеств и математической логики: учеб. пособие / Яросл. гос. ун-т. им. П. Г. Демидова, Ярославль, 2009 — 412 с.

[5] Матиясевич, Ю. В. Алгоритм Тарского // Компьютерные инструменты в образовании. — 2008. — № 6. — С. 14.

[6] Троелсен, Э. Язык программирования C# 7 и платформы .NET и .NET Core / Э. Троелсен, Ф. Джепикс; пер. с англ. и ред. Ю.Н. Артеменко. — 8-е изд. — М.; СПб.: Диалектика, 2020. — 1328 с.

[7] Tarski, A. A Decision Method for Elementary Algebra and Geometry: Prepared for Publication with the Assistance of J.C.C. McKinsey, Santa Monica, Calif.: RAND Corporation, R-109, 1951.

Приложение А

Ссылки на репозитории на GitHub — https://github.com/romarioGI/Coursework2k19-2k20, https://github.com/romarioGI/Coursework2k20-2k21