МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Ярославский государственный университет им. П.Г.Демидова»

Кафедра компьютерной безопасности и математических методов
обработки информации

Курсовая работа

Алгоритм Тарского: описание и реализация

Научный руководитель
профессор, д-р ф.-м.н.
_____ В.Г. Дурнев
«___» _____ 2020 г.

Студент группы КБ-41СО
_____ Р. А. Гибадулин
«___» _____ 2020 г.

Ярославль, 2020 г.

Оглавление

Введение

Пусть $\mathcal{A}$ — формула логики высказываний. Задача: определить, является ли формула $\mathcal{A}$ тождественно истинной. В некотором смысле это «трудная» задача, однако очень просто предложить алгоритм для ее решения, но который будет «не эффективным» — алгоритм Британского музея. 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, описанию и реализации которого посвящена данная работа.

Цели работы: изучить и описать алгоритм Тарского, и реализовать его в виде компьютерной программы.

Задачи:

- Ввести определения, сформулировать и доказать утверждения необходимые для описания алгоритма Тарского;
- Реализовать компьютерную программу, которая по формуле элементарной алгебры без параметров введенной с клавиатуры, строит эквивалентную бескванторную формулу того же языка.

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 практически 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 формально опишем 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, определены 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, некоторые задачи 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 $\overrightarrow{CD} \perp \overrightarrow{AB}$. 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.

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 $\theta(\cdot) = 2$, и $\phi : P \to \mathbb{N}$ такое, что $\phi(<) = 2$, $\phi(>) = 2$ и $\phi(=) = 2$. Из определения отображений θ и ϕ следует, 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 $\mathbb{R}$.

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

Например, теорема о пересечении высот на языке элементарной алгебры записывается так:

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

Нет необходимости формально вводить такие операции как вычитание, деление и возведение в степень по следующим соображениям:

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

1.3 Элиминация 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 $A((Qx)\mathcal{A}) = \mathcal{B}$, 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 $B(\mathcal{A})$ — бескванторная формула, эквивалентная $\mathcal{A}$.

Доказательство. Определим алгоритм B следующим образом:

- Если $\mathcal{A}$ — бескванторная формула, то $B(\mathcal{A}) = \mathcal{A}$;
- Если $\mathcal{A} = (Qx)\mathcal{B}$, то $B(\mathcal{A}) = A((Qx)B(\mathcal{B}))$. Формула $B(\mathcal{B})$ — бескванторная по построению B, следовательно запись $A((Qx)B(\mathcal{B}))$ корректна, при этом $B(\mathcal{B})$ эквивалентна $\mathcal{B}$, следовательно, $(Qx)\mathcal{B}$ 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 $* \in \{\vee, \&, \rightarrow\}$.

  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, и на каждом шаге выход алгоритма — бескванторная эквивалентная формула. ◀

Данное утверждение верно и для других языков логики предикатов.

Таким образом, для элиминации кванторов произвольной формулы достаточно построить алгоритм A и применить описанный в утверждении 1.1 алгоритм B.

1.4 Алгоритм Тарского

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 нас будет интересовать только знак многочлена.

На данном этапе можно считать, что все рассматриваемые многочлены ненулевые, так как знак нулевого многочлена в любой точке определяется тривиальным образом.

Рассмотрим формулу $\mathcal{A} = (Qx)(f(x)\,\rho\,0)$, где Q — квантор, $f(x)$ — многочлен от одной переменной, ρ — предикат. Известно, что многочлены от одной переменной сохраняют свой знак, то есть на интервалах между корнями значение предиката неравенства или равенства с нулем постоянно. Следовательно, чтобы уметь определять знак значения многочлена в произвольной точке, достаточно знать значения многочлена лишь в **конечном** наборе точек — во всех корнях, в каких-то точках между любой парой соседних корней, а также в точках, одна из которых заведомо правее, а другая — левее всех корней. Тогда 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 этих точек.

Рассмотрим формулу $\mathcal{A} = (Qx)(\Phi(x))$, где $\Phi(x)$ — бескванторная формула, которая может содержать вхождения лишь переменной x. Аналогично предыдущему случаю, пусть множество X состоит из корней многочленов, входящих в формулу $\Phi(x)$, и точек, выбранных между парами соседних корней, а также пусть в это множество входит точка, которая правее всех корней, и точка, которая левее всех корней. 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 $\mathcal{A}$.

По теореме Ролля о нуле производной, для любой пары корней x_1, x_2 между ними существует такая точка ξ, что производная многочлена в точке ξ обращается в ноль. Поэтому в качестве точек между корнями можно использовать корни производной этого многочлена.

Аналогично, множество корней многочлена

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

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

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

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	$f_n(x_1)$	...	$f_n(x_j)$	...	$f_n(x_s)$

Так как нас интересуют только знаки многочленов в этих точках, то достаточно записывать 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. Следует из теоремы Коши о нулях непрерывной функции, непрерывности многочленов как функций и упорядоченности точек $x_1, ..., x_s$. ◀

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 1.3. [1] Каждую конечную систему многочленов можно расширить до конечной полунасыщенной системы.

Утверждение 1.4. [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.

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.

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.

Утверждение 1.7. 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 в ней по не убыванию степени.

Рассмотрим подсистему из одного элемента $p_1(x)$. Согласно утверждению 1.6, система $p_1(x)$ является насыщенной, тогда многочлен $p_1(x)$ представляет собой константу, так как его степень меньше единицы, согласно утверждению 1.7. 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, потому что 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	$\varepsilon_{1,1}$	...	$\varepsilon_{1,j}$	...	$\varepsilon_{1,s}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
p_{k-1}	$\varepsilon_{k-1,1}$	...	$\varepsilon_{k-1,j}$	...	$\varepsilon_{k-1,s}$

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	$\varepsilon_{1,1}$	...	$\varepsilon_{1,j}$	...	$\varepsilon_{1,s}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
p_{k-1}	$\varepsilon_{k-1,1}$	...	$\varepsilon_{k-1,j}$	...	$\varepsilon_{k-1,s}$
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 1.8. 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, следовательно, по утверждению 1.8, $\varepsilon_{k,j} = \varepsilon_{t,j}$. 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, что следует из теоремы Коши о нулях непрерывной функции. Для оставшихся клеток рассмотрим случаи.

+	?	+

+	?	0

Тогда вместо символа ? ставится знак +.

0	?	+

0	?	−

В этих случаях ставится знак + или − соответственно.

−	?	0

−	?	−

В этих — знак −. И наконец, в случае

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, по очереди добавляя многочлены;
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.

1.4.6 Пример работы алгоритма

Рассмотрим формулу $(\forall x)(y < 0 \to x^2 > y)$ и построим эквивалентную ей бескванторную 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 $(y < 0 \to x^2 > y)$ истинна. В результате, на выходе алгоритма получим формулу

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

1.5 Теорема Тарского

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

Алгоритм, предложенный А. Тарским в его работе [7], записывался иначе — предельно формально. 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, менее формальное описание алгоритма доступнее для понимания, что, например, упрощает реализацию алгоритма.

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;
- 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;
- Обеспечить минимальное покрытие юнит-тестами;
- 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 LogicLanguageLib, в которой определены три пространства имён:

- в LogicLanguageLib.Alphabet расположены классы для символов алфавита;
- в LogicLanguageLib.Words — классы для термов и формул языка;
- в LogicLanguageLib.IO — класс, решающий задачи системы ввода.

2.1.1 Класс Symbol

Начнём с класса Symbol (рис. 1) — это абстрактный класс, реализующий интерфейс IEquatable<Symbol>. Все классы для символов алфавита языка элементарной алгебры наследники этого класса, то есть класс Symbol — это базовый класс.

Отдельно стоит отметить абстрактное свойство Priority, которое доступно только для чтения. Оно будет использоваться при построении обратной польской записи алгоритмом Дейкстры.

Методы ToString, GetHashCode и EqualsSameType тоже объявлены как абстрактные. Первые два — это стандартные методы для всех объектов, а последний вызывается при сравнении объектов, в методе Equals, когда известно, что сравниваемые объекты являются объектами одного типа.

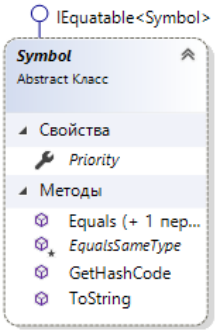

Рис. 1:

Прямыми наследниками класса Symbol являются два абстрактных класса — LogicalSymbol и NonLogicalSymbol (рис. 2), которые описывают логические и нелогические символы соответственно. Оба класса не содержат ни методов, ни свойств, но без них иерархия классов была бы неполной.

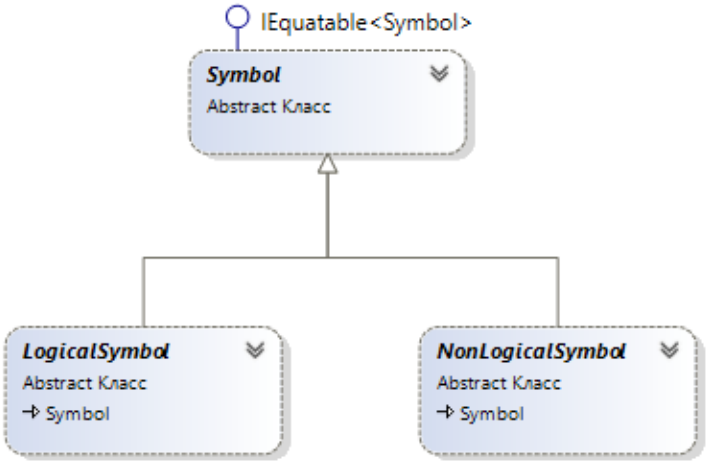

Рис. 2: Фрагмент диаграммы классов.

Далее будут рассмотрены классы для логических символов (рис. 3) и для нелогических символов (рис. 4).

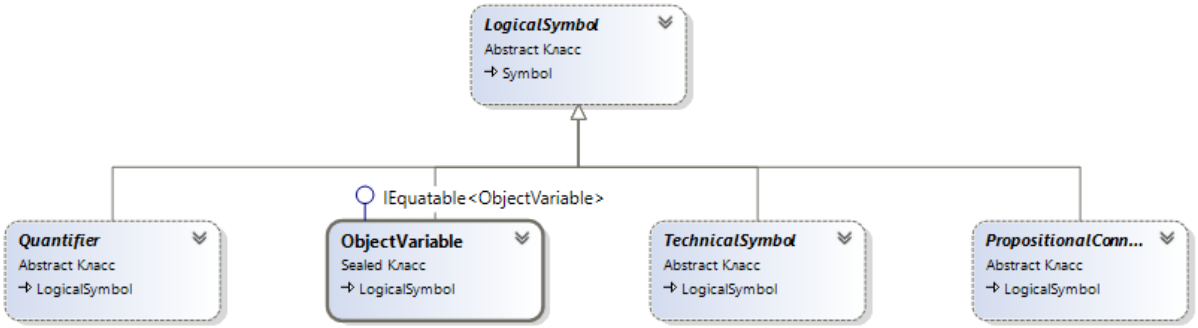

Рис. 3: Фрагмент диаграммы классов.

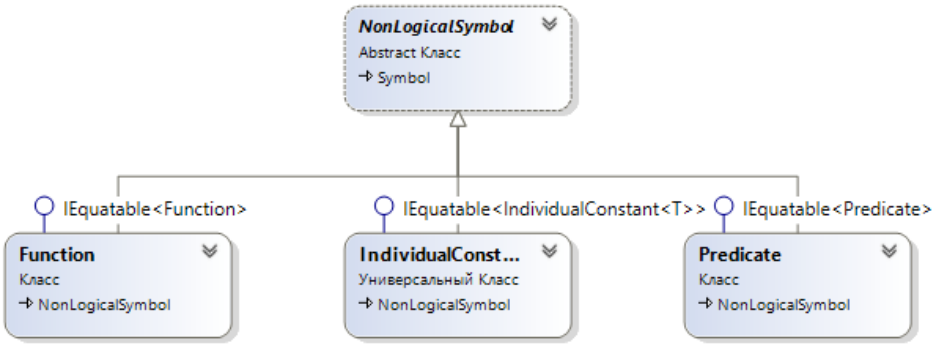

Рис. 4: Фрагмент диаграммы классов.

2.1.2 Класс Quantifier

Класс Quantifier (рис. 5) создан для представления кванторов. Он не содержит никаких специальных методов или свойств, а только лишь реализует свойство Priority — всем кванторам присвоен приоритет 60. Кванторы всеобщности и существования описаны классами производными от класса Quantifier (рис. 6), притом оба класса реализуют паттерн одиночка, что достаточно естественно. Оба класса реализуют метод ToString — метод возвращает строку «∀» или «∃» соответственно.

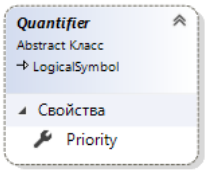

Рис. 5:

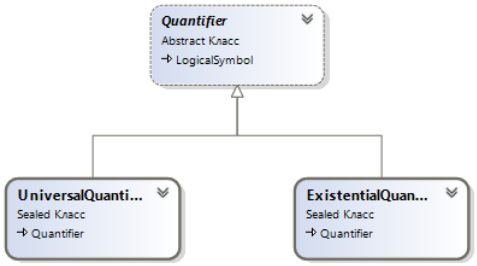

Рис. 6: Фрагмент диаграммы классов.

2.1.3 Класс ObjectVariable

Для индивидных (предметных) переменных написан класс ObjectVariable (рис. 7). Класс реализует интерфейс IEquatable<ObjectVariable>. Каждая переменная — это буква и, возможно, индекс в виде числа. Например, x и A_1 — это переменные, а xx, $a1$, t_i таковыми не являются. Поэтому при создании объекта происходят соответствующие проверки. И две переменные считаются равными, если равны их буквы и индексы.

Приоритет, то есть свойство Priority, для предметных переменных равен -10. А метод ToString, например, для переменной x_1 вернет строку «x_1».

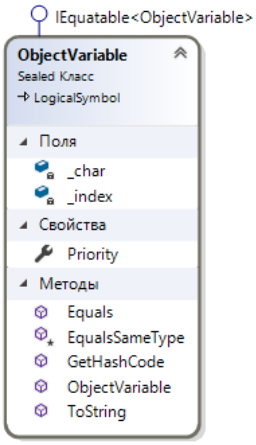

Рис. 7:

2.1.4 Класс TechnicalSymbol

Абстрактный класс TechnicalSymbol не содержит ни одного члена класса, но он является базовым для классов Comma, LeftBracket и RightBracket — запятая, левая и правая скобка соответственно. Классы-наследники вновь реализуют шаблон проектирования одиночка. Приоритет левой скобки равен 0, а правой скобки и запятой равен 10.

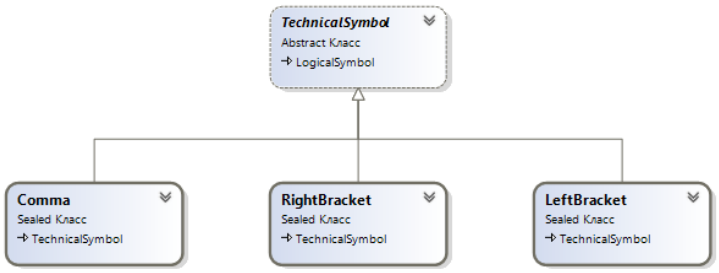

Рис. 8: Фрагмент диаграммы классов.

2.1.5 Класс PropositionalConnective

Эта часть практически полностью взята из предыдущей работы [3], в которой также необходимо было реализовать представление формул, но формул исчисления высказываний.

В абстрактном классе PropositionalConnective (рис. 9) определено поле Arity, то есть арность логической связки.

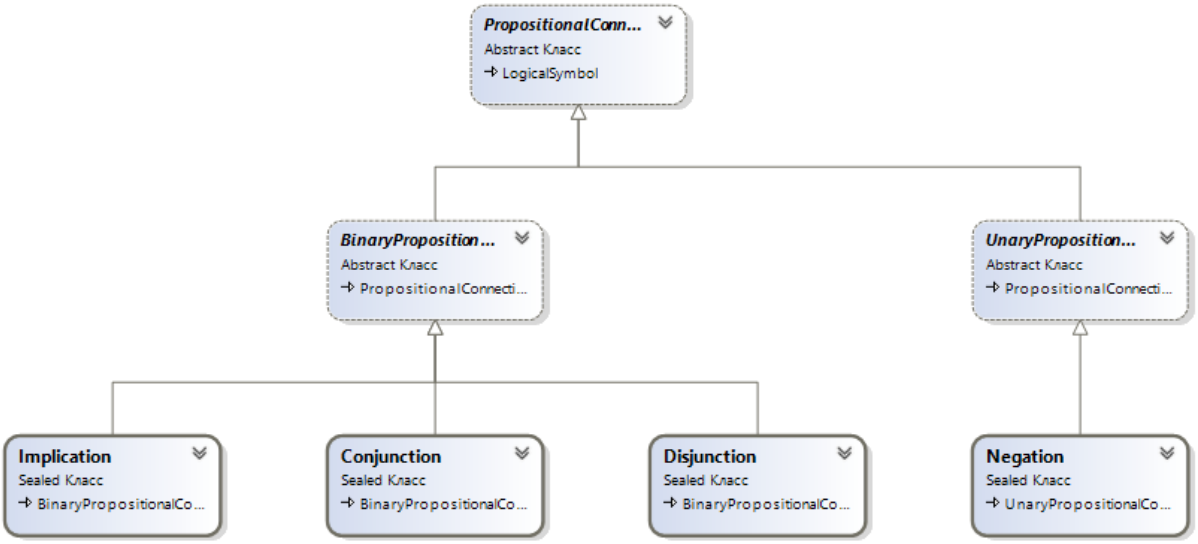

Рис. 9: Фрагмент диаграммы классов.

На диаграмме (рис. 9), под этим классом, расположены ещё два абстрактных класса для бинарных и унарных связок соответственно. В алфавите языка элементарной алгебры три бинарных связки, которые называются импликация, конъюнкция и дизъюнкция, и одна унарная связка — отрицание. Поэтому на нижнем уровне ровно четыре класса. Вновь каждый из этих классов реализует паттерн одиночка, метод ToString возвращает строку с соответствующим классу символом, и определен приоритет — 20, 30, 40 и 50 соответственно для импликации, дизъюнкции, конъюнкции и отрицания.

2.1.6 Класс Function

Перейдем к рассмотрению нелогических символов и начнём с класса Function (рис. 10), то есть с функциональных символов. Среди членов класса отметим два readonly поля _name и Arity. Первое из них хранит имя функции (функциональный символ), а второе — арность этой функции. Данный класс не является абстрактным, поэтому определено значение приоритета, которое равно 100.

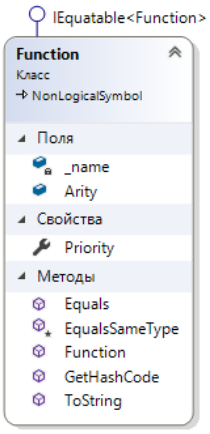

Рис. 10:

На диаграмме (рис. 11) видно, что для бинарных операций сложение, вычитание, умножение, деление и возведение в степень реализованы классы Addition, Subtraction, Multiplication, Division и Exponentiation соответственно. Они не являются прямыми потомка класса Function, а для них создан базовый абстрактный класс ArithmeticBinaryFunction. Каждый класс реализует паттерн одиночка и переопределяет значение приоритета:

- для сложения и вычитания 110,
- для умножения и деления 120,
- и для возведения в степень 140.

А вот прямым потомком класса Function является класс UnaryMinus, который, очевидно из его имени, описывает унарный минус. Он также реализует паттерн одиночка и переопределяет приоритет, который для него равен 130.

Таким образом, реализуется «расширенная версия» языка элементарной алгебры с дополнительными арифметическими операциями, хотя ранее и было отмечено, что это вовсе необязательно.

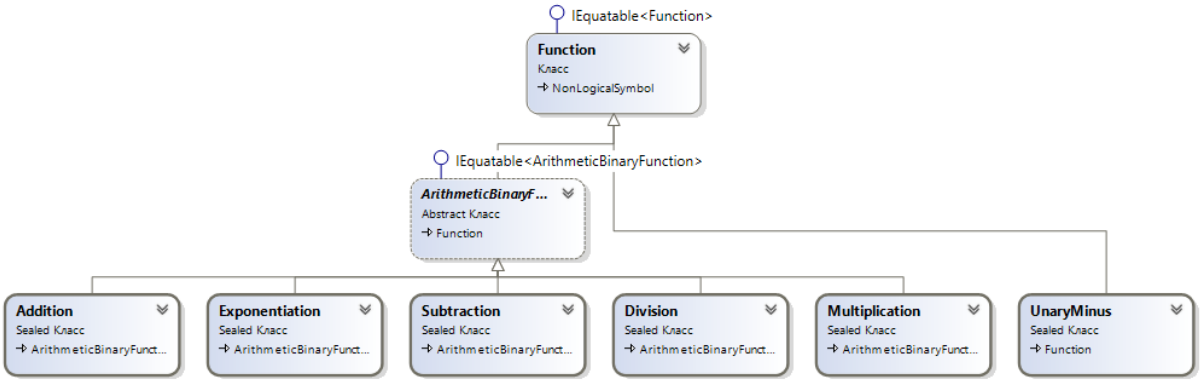

Рис. 11: Фрагмент диаграммы классов.

2.1.7 Класс Predicate

Класс Predicate схож с классом Function. Также определены поля для имени предиката и для его арности, значение свойства Priority равно 70. Ниже на диаграмме (рис. 12) видно два класса-потомка. Оба из них являются абстрактными. Класс ArithmeticPredicate (приоритет равен 80) является базовым для предикатов «меньше», «равно» и «больше» соответственно, а класс BooleanPredicate (приоритет 90) — базовый для **нульместных** предикатов «Ложь» и «Истина».

Формально, местность (арность) предиката — это натуральное число, но можно считать, что нульместные предикаты — это истинностные значения **И** и **Л**.

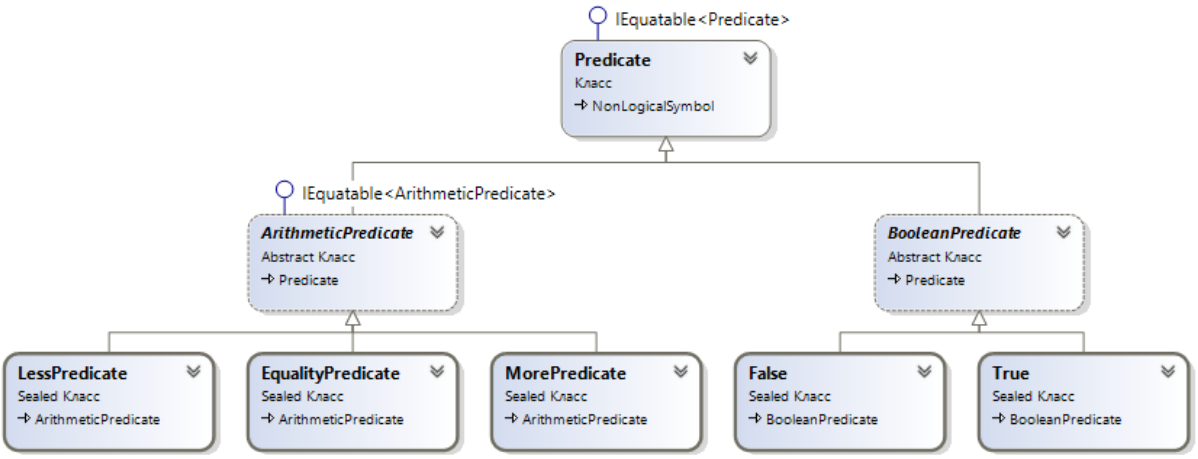

Рис. 12: Фрагмент диаграммы классов.

2.1.8 Класс IndividualConstant

Для констант языка написан обобщенный класс IndividualConstant<T> (рис. 13), который просто оборачивает тип T. Например, IndividualConstant<int> — это целые числа, а рациональные числа записываются как «целое делить на целое». Приоритет констант равен −10.

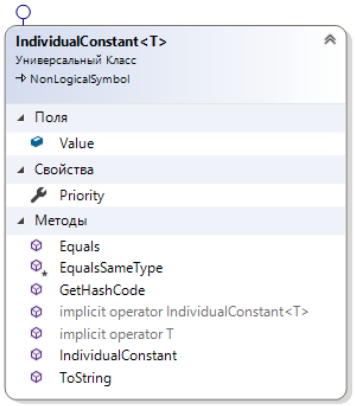

Рис. 13:

На этом описание классов для символов языка закончено.

2.1.9 Класс Term

Прежде чем перейти к описанию формул, необходимо описать класс для термов языка элементарной алгебры. При определении терма выделяется три случая: константа, переменная либо функция от термов. Поэтому абстрактный класс Term является базовым для трех классов (рис. 14). Чтобы для формул находить список свободных переменных, объявлено абстрактное свойство FreeObjectVariables, которое очевидным образом реализуют наследники.

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

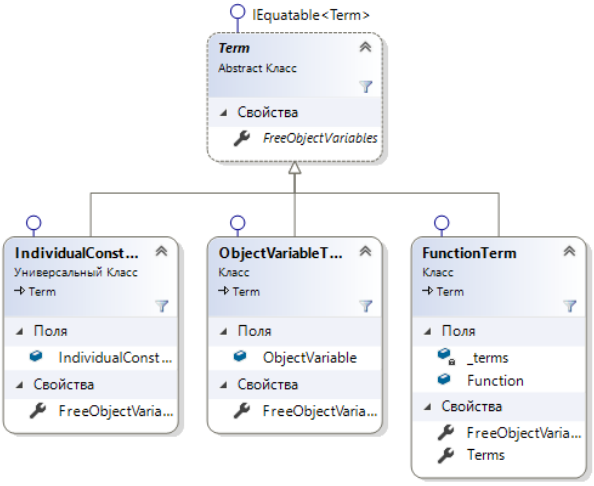

Рис. 14: Фрагмент диаграммы классов.

2.1.10 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 (рис. 15): во главе абстрактный класс Formula и три класса-наследника.

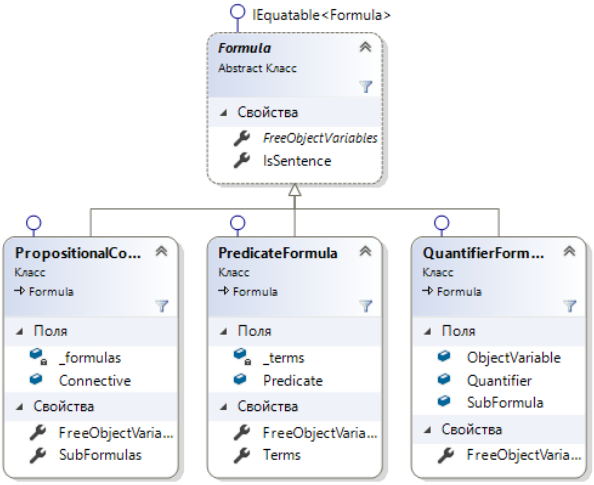

Рис. 15: Фрагмент диаграммы классов.

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]. В результате, был реализован метод ToFormula класса Parser, в работе которого можно выделить три основных этапа:

1. **лексический анализ** — преобразование символов строки в символы алфавита языка элементарной алгебры;

2. построение **обратной польской записи**;

3. и **синтаксический анализ**, в результате которого будет построен экземпляр класса Formula.

Прежде чем перейти к анализу введенной строки, необходимо задать грамматику входного языка:

<формула> → (<формула>) |
<унарная связка><формула>|
<формула><бинарная связка><формула>|
<терм><арифметический предикат><терм>|
<квантор> (<переменная>, <формула>)
<унарная связка> → \lnot
<бинарная связка> → \land|\lor|\to
<арифметический предикат> → <|>|=
<квантор> → \exists|\forall
<переменная> → <буква>|<буква>_<натуральное число>
<терм> → (<терм>) |
<переменная>|
<натуральное число>|
<унарный минус><терм>|
<терм><арифметическая функция><терм>|
<терм><возведение в степень><натуральное число>
<унарный минус> → −
<арифметическая функция> → +| − | ∗ |/|\over
<возведение в степень> → ∧

Нетерминальные символы <натуральное число> и <буква> определяются стандартным образом.

При реализации была заложена возможность вводить функции и предикаты с произвольным именем и с произвольной арностью (местностью), но в грамматике это не отражено, так как данный механизм далее никак не используется.

На этапе лексического анализа входная строка разбивается на терминальные символы (лексемы, токены), в нашем случае, на объекты класса Symbol, пробелы опускаются. Если обнаружен символ, который не принадлежит алфавиту, то генерируется сообщение об ошибке с указанием неопознанного символа и его индекса во входной строке. Все выше перечисленное реализовано в методе ToSymbols класса Parser. А также в этом методе реализован механизм, который отличает унарный минус от бинарного.

Далее, вызовом метода ToRpn, строится обратная польская запись последовательности символов алгоритмом Дейкстры. Этот алгоритм просматривает список входных символов. В зависимости от приоритета символ либо сразу помещается в выходную последовательность (отрицательный приоритет), либо выталкивает из стека

символы с большим приоритетом, после чего сам заносится в стек. При этом метод ToRpn производит ряд простых проверок расположения символов в строке, например, две переменные не могут идти друг за другом.

И наконец, по правилам грамматики «вычисляется» формула, объект класса Formula, таким же способом как происходит вычисление арифметического выражения, записанного в виде обратной польской записи. Именно на этом этапе обнаруживается большинство ошибок, для каждой из которых генерируется сообщение об ошибке.

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. 16) 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.

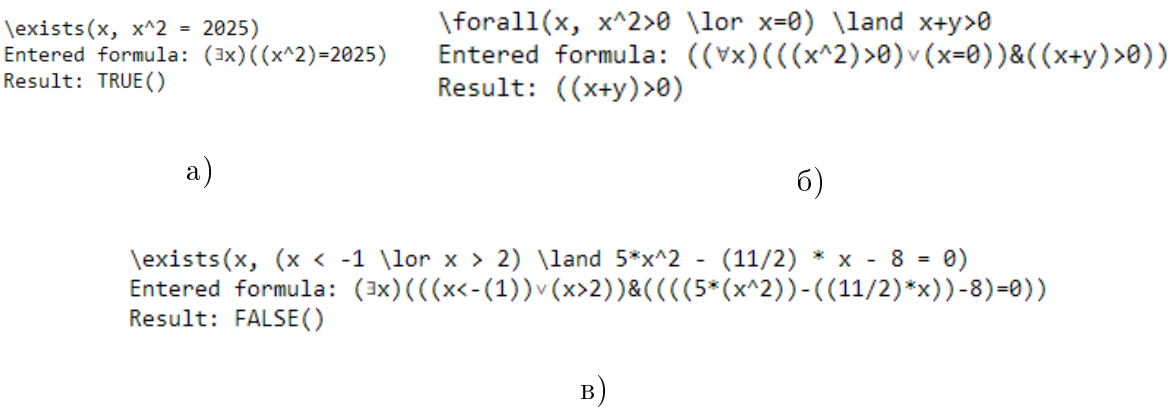

Рис. 16: Примеры выходных данных.

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 квантров. 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

Ссылка на репозиторий на GitHub — https://github.com/romarioGI/Coursework2k19-2k20