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 кафедрой
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 33 страницы, 2 главы, 2 иллюстрации, 18 листингов исходного кода, 1 приложение, в работе использовано 8 источников.

В главе 1 вводятся понятия языка элементарной алгебры, элиминации кванторов, таблицы Тарского, насыщенной системы многочленов. Приводятся методы построения таблиц Тарского, насыщения системы многочленов. Формулируются и доказываются необходимые утверждения про насыщенные системы и таблицы Тарского. Записывается алгоритм Тарского. Формулируется теорема Тарского о языке элементарно алгебры.

В главе 2 обсуждается программная реализация алгоритма Тарского, а также архитектура и конкретные реализации её систем. Обсуждается ввод и вывод формул, лексический и синтаксический анализ формул языка элементарной алгебры. Строится грамматика этого языка. Приводятся примеры результатов работы программы. Говорится о её возможных приложениях.

Ключевые слова: алгоритм Тарского, алгоритмическая разрешимость, элиминация кванторов, логика предикатов, логика высказываний, элементарная алгебра, лексический анализ, синтаксический анализ

Оглавление

Введение

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 [8], 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 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.

2.1 Постановка задач

Для поэтапного создания программы были сформулированы и решены следующие задачи:

- Разработать архитектуру приложения, реализовать функции отдельных его блоков в виде интерфейсов;
- Реализовать систему, решающую задачу получения данных извне, например, из файла или консольного приложения;
- Реализовать систему, решающую задачу преобразования данных в формат, в котором формула будет выводиться экран или сохраняться в файл;
- Реализовать систему лексического и синтаксического анализов формул языка элементарной алгебры;
- Реализовать систему эквивалентных преобразований формул, реализовать алгоритм Тарского как одно из таких преобразований;
- Реализовать консольное приложение, которое применяет алгоритм Тарского к введенной формуле и выводит в консоль результат его работы;
- Покрыть основные модули программы модульными и интеграционными тестами.

2.2 Инструменты разработки

Для реализации алгоритма были выбраны язык C# версии 9.0 и платформа .NET Core 5 [7]. 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;
- Развитые и удобные средства параллельного программирования языка.

Написание программы осуществлялось в среде разработки Rider версии 2020.3. Доступ к программе был получен по студенческой лицензии (рис. 1).

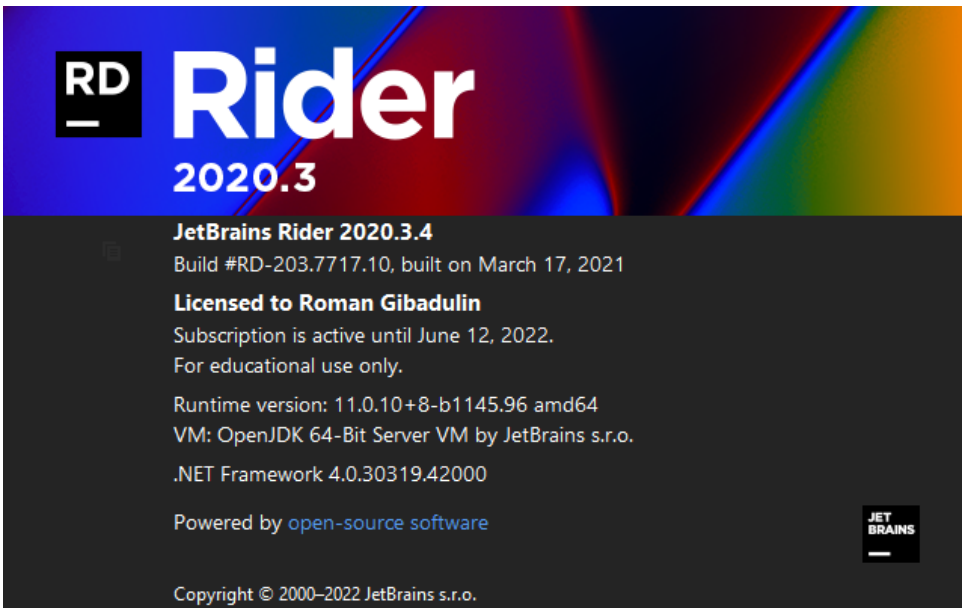

Рис. 1: Информация о среде разработки.

2.3 Архитектура приложения

В результате проделанной работы по проектированию приложения, исходя из предыдущего опыта создания программ, преобразующих логические формулы [3], было решено разделить программу на следующие слабосвязанные системы:

- Система ввода — InputSubsystem.

  Система выполняет функцию получения данных извне, например, из файла или консольного приложения;

- Система вывода — OutputSubsystem.

  Эта система решает задачу преобразования внутреннего формата представления формулы в человекочитаемый формат;

- Система парсинга — ParserSubsystem.

  Данный модуль методами лексического и синтаксического анализов из символьного представления формулы строит объект-формулу, с которой будут работать преобразователи. Также в этом модуле определяется реализация формулы как объекта;

- Система эквивалентных преобразований — ProcessorsSubsystem.

  Данная система определяет реализации процессоров, то есть эквивалентных преобразований формул. В том числе здесь располагается реализация алгоритма Тарского;

- Консольное приложение для демонстрации результатов работы алгоритма Тарского.

  И наконец в этой части программы создано консольное приложение для демонстрации работы программы и алгоритма Тарского.

Взаимодействие между система происходит следующим образом:

- Пользователь вводит с клавиатуры формулу в окне консольного приложения;
- Система ввода считывает эту формулу и передает в систему парсинга в виде последовательности символов;
- Система парсинга по последовательности символов строит объект средствами лексического или синтаксического анализов, представляющий собой формулу языка элементарной алгебры, и который умеют обрабатывать преобразователи;
- Система эквивалентных преобразований применяет процессоры, в том числе, алгоритм Тарского, для построения эквивалентной формулы. Полученная эквивалентная формула передаётся в системы вывода;
- Система вывода преобразует формулу в строковый вид, в котором она выводится в окно консольного приложения пользователю.

Далее рассмотрим каждую систему и её реализацию подробнее, при этом для наглядности будут приведены листинги исходного кода. В них будут отражаться члены классов или интерфейсов, а реализации конкретных методов будут пропускаться.

2.4 Система ввода

Система ввода должна решать получения данных извне, например, из файла или консольного приложения, и передачи эти данных в виде последовательности символов в систему парсинга.

2.4.1 Интерфейсы

Система ввода представлена интерфейсом IInput<T> (листинг 1). Интерфейс является обобщенным и ковариантным. Тип T должен реализовывать интерфейс ISymbol, который не содержит ни одного метода и ни одного свойства. Интерфейс содержит один метод MoveNext, который пытается перейти к следующему символу во входных данных и возвращает сигнал об успешности этого действия, то есть этот метод аналогичен одноименному методу интерфейса IEnumerator. Также интерфейс определяет два доступных только для чтения свойства: Current и IsOver. Первое возвращает текущий просматриваемый элемент входной последовательности. Второй истинно, если чтение входной последовательности завершено, иначе ложно.

Таким образом, интерфейс позволяет последовательно получать символы введенные пользователем в виде объектов типа T.

```
namespace InputSubsystem
{
    public interface IInput<out T> where T: ISymbol
    {
        public bool MoveNext();

        public T Current { get; }

        public bool IsOver { get; }
    }
}
```

Листинг 1: Интерфейс IInput<T>

2.4.2 Реализация

Прежде чем перейти к реализации интерфейса IInput, предложим реализацию интерфейса ISymbol — класс Symbol (листинг 2). Он предоставляет информацию о непосредственно символе и о его порядковом номере во входных данных.

```
namespace InputSubsystem
{
    public class Symbol : ISymbol
    {
        public readonly int Index;
        public readonly char Character;

        public Symbol(int index, char character)
        {
        }
    }
}
```

Листинг 2: Класс Symbol

Интерфейс реализован классом TextReaderInput (листинг 3). Можно сказать, что класс является оберткой над классом System.IO.TextReaderInput. Во-первых, конструктор требует всего один аргумент типа TextReaderInput. В конструкторе инициализируются приватные поля класса и свойства интерфейса. Во-вторых, метод MoveNext при каждом вызове пытается прочитать очередной символ из _textReader. Если прочитать символ не получилось или прочитанный символ является символом перевода конца строки, то чтение прекращается. Иначе в Current помещается новый объекта типа Symbol, полученный из прочитанного символа и текущего числа символов, счетчик символов _symbolNumber увеличивается на один.

```
namespace InputSubsystem
{
    public class TextReaderInput : IInput<Symbol>
    {
        private readonly TextReader _textReader;
        private int _symbolNumber;

        public TextReaderInput(TextReader textReader)
        {
        }

        public bool MoveNext()
        {
        }

        public Symbol Current { get; private set; }

        public bool IsOver { get; private set; }
    }
}
```

Листинг 3: Класс TextReaderInput

Непосредственно для ввода данных с консоли реализован класс ConsoleInput, который является классом наследником класса TextReaderInput (листинг 4).

```
namespace InputSubsystem
{
    public class ConsoleInput: TextReaderInput
    {
        public ConsoleInput() : base(Console.In)
        {
        }
    }
}
```

Листинг 4: Класс ConsoleInput

А для чтения данных из файла — класс FileInput (листинг 5).

```
namespace InputSubsystem
{
    public class FileInput : TextReaderInput
    {
        private static TextReader GetFileTextReader(string
           fileName)
        {
            return new StreamReader(fileName);
        }

        public FileInput(string fileName) :
           base(GetFileTextReader(fileName))
        {
        }
    }
}
```

Листинг 5: Класс FileInput

2.5 Система парсинга

Система парсинга решает задачу лексического и синтаксического анализов формулы языка элементарной алгебры, а также определяет внутреннее представление формулы в виде объекта.

2.5.1 Интерфейсы

Объекты, определяющие внутреннее представление формулы языка элементарной алгебры, должны реализовывать интерфейс IExpression (листинг 6). Единственный член этого интерфейса — readonly-свойство Type типа ExpressionType, которое определяет тип выражения: формула, терм или идентификатор, то есть литерал.

ExpressionType — перечисление (enum), которое имеет лишь три значения (листинг 7). Эти значения несут информацию о том, какой тип имеет данное выражение языка: формула, терм или идентификатор соответственно.

```
namespace ParserSubsystem
{
    public interface IExpression
    {
        public ExpressionType Type { get; }
    }
}
```

Листинг 6: Интерфейс IExpression

```
namespace ParserSubsystem
{
    public enum ExpressionType
    {
        Formula,
        Term,
        Identifier
    }
}
```

Листинг 7: Перечисление ExpressionType

Следующая функция, которую выполняет система парсинга — построение внутреннего представления формулы по входной последовательности символов. Данная функция возложена на интерфейс IParser<TS, TE> — обобщенный ковариантный по первому, контравариантный по второму типу интерфейс (листинг 8). Первый тип должен реализовывать интерфейс ISymbol, второй — IExpression. Интерфейс определяет всего один метод Parse, который по входным данным типа IInput<TS>, на выходе отдает объект типа TE. Иначе говоря, эта функция преобразует введенную строку во внутреннее представление логических выражение.

```
namespace ParserSubsystem
{
    public interface IParser<in TS, out TE> where TS:
       ISymbol where TE: IExpression
    {
        public TE Parse(IInput<TS> input);
    }
}
```

Листинг 8: Интерфейс IParser

2.5.2 Реализация

Интерфейс IExpression реализует класс SyntaxTree (листинг 9). Класс содержит readonly поле Token типа Token, три readonly свойства OperandsCount, Operands, Type, один метод и единственный конструктор. Из имени класса очевидно, что для внутреннего представления формул используется дерево синтаксического разбора. Пройдемся по членам этого класса.

В дереве синтаксического разбора каждая вершина отмечена терминальным символом, которые мы будем называть токенами. Класс Token — абстрактный класс, являющийся классом наследником класса Word, новых членов по сравнению с родительским классом не добавляет (листинг 10). В свою очередь класс Word также является абстрактным, абстрактно переопределяет метод ToString (листинг 11).

```
namespace ParserSubsystem
{
    public class SyntaxTree : IExpression
    {
        public readonly Token Token;

        public int OperandsCount => Operands.Length;

        public ImmutableArray<SyntaxTree> Operands { get; }

        public SyntaxTree GetOperand(int index)
        {
        }

        public SyntaxTree(ExpressionType type, Token token,
           params SyntaxTree[] operands)
        {
        }

        public ExpressionType Type { get; }
    }
}
```

Листинг 9: Класс SyntaxTree

```
namespace ParserSubsystem
{
    public abstract class Token : Word
    {
        public abstract override string ToString();
    }
}
```

Листинг 10: Класс Token

```
namespace ParserSubsystem
{
    public abstract class Word
    {
        public abstract override string ToString();
    }
}
```

Листинг 11: Класс Word

Свойство OperandsCount, очевидно из названия, определяет количество операндов, то есть, количество дочерних вершин. Сами операнды расположены в неизменяемом массиве Operands, элементы которого также имеют тип SyntaxTree. Метод GetOperand возвращает операнд по его номеру.

В классе SyntaxTree определен единственный конструктор, принимающий на вход тип выражения, токен и массив операндов.

И последний член этого класса — Type, реализует соответствующее свойство интерфейса.

Переходим к классу SyntaxTreeParser, который реализует интерфейс IParser, а точнее IParser<Symbol, SyntaxTree> (листинг 12).

```
namespace ParserSubsystem
{
    public class SyntaxTreeParser : IParser<Symbol,
       SyntaxTree>
    {
        public SyntaxTree Parse(IInput<Symbol> input)
        {
            var lexemes =
               ToLexemes(input).ToImmutableArray();
            var syntaxTree = ToSyntaxTree(lexemes);

            return syntaxTree;
        }

        private static IEnumerable<Lexeme>
           ToLexemes(IInput<Symbol> input)
        {
        }
        private static SyntaxTree
           ToSyntaxTree(ImmutableArray<Lexeme> lexemes)
        {
        }
    }
}
```

Листинг 12: Класс SyntaxTreeParser

Верхнеуровнево, метод Parse реализован следующим образом: по последовательности входных символом строится последовательность лексем методами лексического анализа, затем из последовательности лексем получается синтаксическое дерево методами синтаксического анализа. Теперь подробнее разберем как проводится разбиение на лексемы и каким образом строится синтаксическое дерево.

Прежде чем перейти к этапу лексического анализа, необходимо сказать о классе Lexeme (листинг 13). Этот класс является абстрактным, абстрактно переопределяет метод ToString родительского класса Word, а также содержит два readonly свойства FirstSymbolIndex и LastSymbolIndex, которые хранят информацию об индексе относительно входных данных первого и последнего символа, образующих лексему.

```
namespace ParserSubsystem
{
    public abstract class Lexeme : Word
    {
        public abstract int FirstSymbolIndex { get; }
        public abstract int LastSymbolIndex { get; }

        public abstract override string ToString();
    }
}
```

Листинг 13: Класс Lexeme

Конкретными реализациями абстрактного класса Lexeme являются классы LiteralLexeme и SpecialSymbolLexeme. Первый определяет лексемы состоящие только лишь из букв и символов, то есть литералы. Эти последовательности также могут начинаться с символа обратного слэша. Второй — остальные символы, кроме обратного слэша. В то время как LiteralLexeme хранит информацию о литерале в виде строки, класс SpecialSymbolLexeme хранит информацию лишь об одном символе типа Symbol.

Разбиение на лексемы производится достаточно стандартным образом:

- пропускаются все пробельные символы, они лишь являются разделителями;
- последовательности из чисел и букв, возможно, начинающиеся с символа обратного слэша, образуют лексемы-литералы;
- остальные символы — лексемы-спецсимволы.

Напомним, что реализован синтаксический разбор в методе ToSyntaxTree. Если для языка элементарной алгебры построить контекстно-свободную грамматику, то методом рекурсивного нисходящего разбора[6] можно построить дерево разбора. Реализовать этот метод нетрудно, нужно лишь научиться сопоставлять лексемы токенам, и для каждой группы правил с одинаковой левой частью написать функцию, которая последовательно пытается применить каждое из правил. Также эти функции, подобно машине Тьюринга, должны иметь возможность обозревать последовательность лексем, перемещаясь по этой последовательность вправо или влево. Каждая из этих функций в нашей реализации на вход принимает параметр типа ParsingContext, который позволяет просматривать последовательность лексем, разбивать ее на подпоследовательности, а также реализует подсчет баланса скобок. А возвращают эти функции одну из реализаций абстрактного класса Token: IdentifierToken или OperatorToken. Токены первого типа — это константы или переменные, а второго — остальные символы алфавита языка элементарной алгебры, а также два новых символа для нульместных предикатов (листинг 14).

```
OperatorName.ExistentialQuantifier => "∃",
OperatorName.UniversalQuantifier => "∀",
OperatorName.Conjunction => "&",
OperatorName.Disjunction => "∨",
OperatorName.Implication => "→",
OperatorName.Negation => "¬",
OperatorName.Less => "<",
OperatorName.More => ">",
OperatorName.Equal => "=",
OperatorName.Plus => "+",
OperatorName.Minus => "-",
OperatorName.Multi => "*",
OperatorName.Divide => "/",
OperatorName.Exponentiation => "^",
OperatorName.True => @"\true",
OperatorName.False => @"\false",
```

Листинг 14: Соответствие OperatorName и строкового представления

При этом можно реализовать ряд эвристик, чтобы уменьшить временную сложность алгоритма нисходящего разбора вплоть до линейной или квадратичной. Например, можно использовать понятие баланса скобок, благодаря чему находить

ведущий операнд. В нашей реализации была достигнута именно квадратичная от длины входной строки временная сложность, чего с запасом достаточно для нашей задачи. Действительно, алгоритм Тарского имеет сверх экспоненциальную временную сложность, поэтому быстрая обработка длинных строк неактуальна в нашем случае, хотя и возможна.

И наконец, выпишем в форме Бэкуса–Наура[6] грамматику языка элементарной алгебры. В этой грамматике учтено, что возведение в степень обладает правой ассоциативностью, в то время как остальные бинарные операции лево-ассоциативны. Также в эту грамматику заложен приоритет арифметических операций.

⟨*формула*⟩ ::= ⟨*кванторная ф-а*⟩

⟨*кванторная ф-а*⟩ ::= (⟨*квантор*⟩⟨*переменная*⟩)⟨*кванторная ф-а*⟩
| ⟨*ф-а с бин. связкой*⟩

⟨*ф-а с бин. связкой*⟩ ::= ⟨*ф-а с бин. связкой*⟩⟨*бин. связка*⟩⟨*ф-а с ун. связкой*⟩
| ⟨*ф-а с ун. связкой*⟩

⟨*ф-а с ун. связкой*⟩ ::= ⟨*унарная связка*⟩⟨*ф-а с ун. связкой*⟩
| ⟨*предикатная ф-а*⟩

⟨*предикатная ф-а*⟩ ::= ⟨*терм*⟩⟨*предикат*⟩⟨*терм*⟩
| (⟨*формула*⟩)

⟨*квантор*⟩ ::= \exists
| \forall

⟨*бин. связка*⟩ ::= \land
| \lor
| \to

⟨*унарная связка*⟩ ::= \not

⟨*предикат*⟩ ::= $>$
| $<$
| $=$

⟨*терм*⟩ ::= ⟨*+- терм*⟩

⟨*+- терм*⟩ ::= ⟨*+- терм*⟩⟨*плюс минус*⟩⟨**/ терм*⟩
| ⟨**/ терм*⟩

⟨*плюс минус*⟩ ::= $+$
| -

⟨**/ терм*⟩ ::= ⟨**/ терм*⟩⟨*умножение деление*⟩⟨*exp терм*⟩
| ⟨*exp терм*⟩

⟨*умножение деление*⟩ ::= $*$
| $/$
| \over

⟨*exp терм*⟩ ::= ⟨*- терм*⟩⟨*возв. в степень*⟩⟨*exp терм*⟩
| ⟨*- терм*⟩

⟨*возв. в степень*⟩ ::= ^

⟨- *терм*⟩ ::= ⟨*минус*⟩⟨- *терм*⟩
| ⟨*литерал*⟩

⟨*минус*⟩ ::= -

⟨*литерал*⟩ ::= ⟨*переменная*⟩
| ⟨*нат. число*⟩
| (⟨*терм*⟩)

⟨*переменная*⟩ ::= ⟨*буква*⟩
| ⟨*буква*⟩_⟨*нат. число*⟩

⟨*буква*⟩ ::= а | б | ... | я
| А | Б | ... | Я
| a | b | ... | z
| A | B | ... | Z

⟨*нат. число*⟩ ::= ⟨*цифра*⟩
| ⟨*цифра*⟩⟨*нат. число*⟩

⟨*цифра*⟩ ::= 0 | 1 | ... | 9

Если в ходе разбора не удалось применить ни одно из правил, то бросается исключение с информацией об возможной ошибке, допущенной при вводе формулы.

2.6 Система вывода

Система вывода решает задачу преобразования данных в формат, в котором формула будет выводиться экран или сохраняться в файл.

2.6.1 Интерфейс

Возложенная на систему вывода функцию описывает обобщенный контравариантный интерфейс IOutput<TE>, где тип TE должен реализовывать интерфейс IExpression (листинг 15). Интерфейс содержит единственный метод Print, который по выражению строит его строковое представление.

```
namespace OutputSubsystem
{
    public interface IOutput<in TE> where TE : IExpression
    {
        public string Print(TE expression);
    }
}
```

Листинг 15: Интерфейс IOutput

2.6.2 Реализация

Интерфейс IOutput<SyntaxTree> реализован классом NativeOutput. Не будем вдаваться в подробности реализации этого класса. Но отметим, что он преобразует формулу в строку учитывая приоритет операций, их арность, нотацию записи и левую или правую ассоциативность.

Для вывода кванторов, пропозициональных связок и других особых символов используются соответствующее символы юникода (листинг 14). Отметим, что метод расставляет скобки с учетом приоритета операций, то есть лишние скобки не ставятся. Эта достигается подсчетом наименьшего приоритета операций, свободных от скобок, в поддереве обозреваемого оператора.

2.7 Система эквивалентных преобразований

В этой системе определены реализации эквивалентных преобразований формул, в том числе, алгоритма Тарского.

2.7.1 Интерфейс

Все процессоры должны реализовывать интерфейс IProcessor<T>, где тип T в свою очередь реализует интерфейс IExpression (листинг 16). В интерфейсе определен лишь один метод, который должен преобразовать одно выражение в другое.

```
namespace ProcessorsSubsystem
{
    public interface IProcessor<T> where T: IExpression
    {
        public T Do(T expression);
    }
}
```

Листинг 16: Интерфейс IProcessor

```
namespace ProcessorsSubsystem
{
    public abstract class SyntaxTreeProcessor :
       IProcessor<SyntaxTree>
    {
        protected abstract SyntaxTree DoInner(SyntaxTree
           syntaxTree, bool recursively);

        public SyntaxTree Do(SyntaxTree syntaxTree)
        {
            return DoInner(syntaxTree, true);
        }

        public SyntaxTree DoOnlyRoot(SyntaxTree syntaxTree)
        {
            return DoInner(syntaxTree, false);
        }
    }
}
```

Листинг 17: Класс SyntaxTreeProcessor

А все преобразователи деревьев разбора наследуются от абстрактного класса SyntaxTreeProcessor, который реализует интерфейс IProcessor<SyntaxTree> (листинг 17).

2.7.2 Реализация

Первый преобразователь — ZeroArityPredicateEliminator. Его задача заключается в том, чтобы сократить формулу используя свойства формул логики высказываний. Например, если в конъюнкции хотя бы один элемент есть нульместный предикат ложь, то и вся конъюнкция принимает значение ложь. А чтобы применить все эти правила, метод совершает обход в глубину дерева разбора, при этом обход поддеревьев производится параллельно, в несколько потоков.

Главный преобразователь, ради которого и разрабатывалась вся программа — TarskiQuantifierEliminator. Вновь описывать алгоритм Тарского мы не станем. Также не станем повторять алгоритм насыщения системы многочленов, который реализован классом Saturator.

Укажем, что таблица Тарского, реализованная классом TarskiTable, представляет собой связный список столбцов, где столбцы — это списки элементов типа Sign. В этот класс скрыты все шаги по добавлению нового многочлена в таблицу. Очевидно, что столбцы хранятся в связном списке потому, что нам необходимо уметь эффективно добавлять новый столбец в любое место таблицы, а с этой задачей лучше всех справляется связный список. Сами столбцы являются списками, так как происходит только добавление в конец столбца, при этом нужно иметь доступ к любому элементу столбца по его индексу, с чем лучше всех справляется List.

А сами многочлены были реализованы стандартным образом — массив коэффициентов типа RationalNumber, где реализация рационального числа также очевидна и не нуждается в комментариях.

2.8 Тестирование

Тесты расположены в проекте Tests и разделены на два типа: модульные и интеграционные тесты. Первые контролируют работоспособность таблиц Тарского, «сатуратора» и многочленов. Вторые — системы ввода, парсинга и вывода в связке.

Всего написано 27 тестов, из которых 22 модульные, оставшиеся 5 интеграционные.

2.9 Примеры результатов работы программы

Ниже приведены результаты запусков консольного приложения и его исходный код, которое инициализирует все системы и последовательно их запускает (листинг 18).

```
namespace TarskiAlgorithmConsoleApp
{
    internal static class Program
    {
        private static readonly IInput<Symbol> Input = new
            ConsoleInput();
        private static readonly IParser<Symbol, SyntaxTree>
            Parser = new SyntaxTreeParser();
```

```
        private static readonly IProcessor<SyntaxTree>
            TarskiProcessor = new
            TarskiQuantifierEliminator();
        private static readonly IOutput<SyntaxTree> Output =
            new NativeOutput();

        private static void Main()
        {
            try
            {
                Console.OutputEncoding =
                    System.Text.Encoding.UTF8;
                Console.Clear();

                Console.WriteLine("Please, enter formula");

                var parsedInput = Parser.Parse(Input);
                Console.WriteLine($"Input
                    formula:\t{Output.Print(parsedInput)}");

                var eliminatedFormula =
                    TarskiProcessor.Do(parsedInput);

                Console.WriteLine($"Result
                    formula:\t{Output.Print(eliminatedFormula)}");
            }
            catch (Exception e)
            {
                Console.WriteLine($"ERROR: {e.Message}");
                throw;
            }
            Console.WriteLine("Press any key to shut down
                the program");
            Console.ReadKey();
        }
    }
}
```

Листинг 18: Консольное приложение

```
Please, enter formula
(\forall x) x=x
Input formula:  (∀x)x=x
Result formula: \true
```

а)

```
Please, enter formula
(\exists x) x^2 - 2*x + 1 = 0 \land x = 0
Input formula:  (∃x)x^2-2*x+1=0&x=0
Result formula: \false
Press any key to shut down the program
```

б)

```
Please, enter formula
((\exists x) x^2 - 2*x + 1 = 0) \land x=0
Input formula:  ((∃x)x^2-2*x+1=0)&x=0
Result formula: x=0
Press any key to shut down the program
```

в)

```
Please, enter formula
(\forall a) a^10 + 1 > 0 \land ((\exists b) b^3 + 10 = 0)
Input formula:  (∀a)a^10+1>0&((∃b)b^3+10=0)
Result formula: \true
```

г)

2.10 О возможных приложениях

Несколько слов скажем о возможных приложениях написанной нами программы.

Во-первых, благодаря гибкой архитектуре, слабым связям между модулями через интерфейсы, данный программный комплекс можно расширять, добавлять в него новые правила грамматики, новый преобразователи формул, тем самым расширяя язык и способы эквивалентных преобразований.

Во-вторых, идет активное развитие систем автоматической верификации программ и моделей безопасности. Если сформулировать требования к их безопасности на формальном языке, то примерно теми же методами можно доказывать их соответствие заявленным требованиям и безопасность.

Заключение

Таким образом, цели данной работы достигнуты, все поставленные задачи решены. В работе описан языка элементарной алгебры, приведены примеры задач элементарной алгебры, достаточно подробно описан алгоритм элиминации кванторов этого языка — алгоритм Тарского.

Алгоритм был реализован как часть программного комплекса. 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.

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] Соколов, В. А. Введение в теорию формальных языков : учебное пособие / Яросл. гос. ун-т им. П. Г. Демидова. — Ярославль : ЯрГУ, 2014. — 208 с.

[7] 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.

[8] 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 с исходным кодом программы — https://github.com/romarioGI/diploma