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 поиска вывода в Исчислении Высказываний и его программная реализация.

Научный руководитель
профессор, д-р ф.-м.н.
_____ В.Г. Дурнев
«___» _____ 2019 г.

Студент группы КБ-31СО
_____ Р. А. Гибадулин
«___» _____ 2019 г.

Ярославль, 2019 г.

Оглавление

Введение

14 сентября 2018 года на лекции по математической логике преподаватель Дурнев Валерий Георгиевич предложил студентам, в том числе и автору этой работы, следующую задачу:

Дано слово ω. Является ли ω формулой Исчисления Высказываний?

Очевидно, что решением этой задачи является некоторый общий метод, то есть алгоритм, поэтому предлагалось написать компьютерную программу решающую эту задачу. Меня заинтересовала эта и другие похожие задачи, предложенные в ходе лекций по математической логике, в которых требовалось написать программу работающую с объектами Исчисления Высказываний. Поэтому уже в качестве курсовой работы мне была предложена задача:

Написать компьютерную программу, которая находит вывод формулы Исчисления Высказываний.

Цель работы: написать компьютерную программу, которая по вводимой с клавиатуры выводимой формуле Исчисления Высказываний находит её вывод в Исчислении Высказываний.

Задачи:

- Разработать и реализовать метод, который по строке символов определяет, является ли она формулой Исчисления Высказываний.
- Разработать и реализовать способ представления формул Исчисления Высказываний в памяти компьютера.
- Разработать и реализовать алгоритм, который проверяет, можно ли добавить формулу в данный вывод в Исчислении Высказываний.
- Разработать и реализовать алгоритм, который по заданной выводимой формуле строит её вывод в Исчислении Высказываний.
- Собрать все алгоритмы в одну программу с удобным в использовании пользовательским интерфейсом.
- С помощью написанной программы, найти выводы для некоторых формул Исчисления Высказываний.

В первой главе будет дано определение Исчисления Высказываний, доказана теорема о количестве подформул формулы, дано определение вывода в Исчислении Высказываний.

Во второй главе будет дано описание алгоритма поиска вывода в Исчислении Высказываний, а также будет дано представление о трудоёмкости алгоритма.

В третей главе пойдет речь о реализации компьютерной программы, которая содержит в себе решение всех поставленных задач.

1. Исчисление Высказываний

В этой главе будет рассмотрен один из простейших языков математической логики – язык **Исчисления Высказывания**. Обозначения и большая часть определений совпадают с теми, что используются в книге [1].

1.1 Язык Исчисления Высказываний

Для того, чтобы задать логический язык L, необходимо определить алфавит языка Σ, а также обозначить множество формул F – подмножество множества всех выражений языка L.

Определение 1. Алфавит $\Sigma_{\text{ИВ}}$ языка $L_{\text{ИВ}}$ Исчисления Высказываний является объединением трех множеств символов: Σ_1, Σ_2, Σ_3, то есть $\Sigma_{\text{ИВ}} = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$. Множество $\Sigma_1 = \{A_1, A_2, \ldots\}$ – счетное множество **пропозициональных переменных**. Множество $\Sigma_2 = \{\neg, \vee, \&, \rightarrow\}$ – конечное множество **пропозициональных связок**. Символ $\neg$ называется отрицанием, символ $\vee$ – дизъюнкцией, символ $\&$ – конъюнкцией, а символ $\rightarrow$ - импликацией. А множество Σ_3 – множество **технических символов**, состоит из двух символов: (- левая скобка, и) – правая скобка.

Определение 2. Формула языка $L_{\text{ИВ}}$ определяется следующим образом:

1. каждая пропозициональная переменная A_i – формула языка $L_{\text{ИВ}}$. Такие формулы называются **элементарными формулами**;

2. если $\mathcal{A}$ и $\mathcal{B}$ – формулы языка $L_{\text{ИВ}}$, то и выражения:

$$(\neg\mathcal{A}),\ (\mathcal{A} \vee \mathcal{B}),\ (\mathcal{A}\&\mathcal{B}),\ (\mathcal{A} \rightarrow \mathcal{B}).$$

   являются формулами языка $L_{\text{ИВ}}$.

Количество символов в формуле (или длину формулы) будем обозначать $|\mathcal{A}|$. А также в работе будут активно использоваться следующие определения:

Определение 3. Формулы $\mathcal{A}$ и $\mathcal{B}$ языка $L_{\text{ИВ}}$ называются **равными**, если они равны как слова языка $L_{\text{ИВ}}$.

Определение 4. Заменой переменных $A_1, A_2, , ..., A_n$ на формулы $\mathcal{B}_1, \mathcal{B}_2, , ..., \mathcal{B}_n$ в формуле $\mathcal{A}$ называется одновременная замена всех вхождений A_1, A_2, , ..., A_n в слово $\mathcal{A}$ соответственно на слова $\mathcal{B}_1$, $\mathcal{B}_2$, , ..., $\mathcal{B}_n$. В результате замены переменных получается формула $\mathcal{A}' = \mathcal{A}_{A_1, A_2, , ..., A_n}[\mathcal{B}_1, \mathcal{B}_2, , ..., \mathcal{B}_n]$.

Определение 5. Множество $SubF_{\mathcal{A}}$ подформул формулы $\mathcal{A}$ языка $L_{\text{ИВ}}$ определяется тремя пунктами:

1. $\mathcal{A} \in SubF_{\mathcal{A}}$;

2. если формула $\mathcal{A}$ имеет вид $(\neg\mathcal{B})$, то $SubF_{\mathcal{B}} \subset SubF_{\mathcal{A}}$;

3. если формула имеет вид $(\mathcal{B} \vee \mathcal{C})$, или $(\mathcal{B}\&\mathcal{C})$, или $(\mathcal{B} \rightarrow \mathcal{C})$, то $SubF_{\mathcal{B}} \subset SubF_{\mathcal{A}}$ и $SubF_{\mathcal{C}} \subset SubF_{\mathcal{A}}$.

Теорема 1. Количество различных подформул формулы Исчисления Высказываний не превосходит количества символов в этой формуле.

Доказательство. Рассмотрим формулу $\mathcal{A}$.

Если формула $\mathcal{A}$ - есть элементарная формула, то количество символов в формуле равно единице, и при этом, по определению, имеется всего одна подформула, которая совпадает с самой формулой. Таким образом, для формул длины равной единице теорема доказана.

Пусть теорема доказана для всех формул, длина которых не превышает n, где $n \in \mathbb{N}$. Пусть $\mathcal{A}$ содержит n символов и $n \neq 1$. Значит, формула $\mathcal{A}$ либо имеет вид $(\neg\mathcal{B})$, тогда имеем

$$\left.\begin{array}{c} \overline{\overline{SubF_{\mathcal{A}}}} = 1 + \overline{\overline{SubF_{\mathcal{B}}}} \\ |\mathcal{A}| = 3 + |\mathcal{B}| \\ |\mathcal{B}| < n \Rightarrow \overline{\overline{SubF_{\mathcal{B}}}} \leq |\mathcal{B}| \end{array}\right\} \Rightarrow \overline{\overline{SubF_{\mathcal{A}}}} \leq 1 + |\mathcal{B}| = |\mathcal{A}| - 2 \leq |\mathcal{A}| \Rightarrow \overline{\overline{SubF_{\mathcal{A}}}} \leq |\mathcal{A}|\,,$$

либо имеет один из видов $(\mathcal{B} \vee \mathcal{C})$, $(\mathcal{B}\&\mathcal{C})$, $(\mathcal{B} \to \mathcal{C})$, тогда

$$\left.\begin{array}{c} \overline{\overline{SubF_{\mathcal{A}}}} \leq 1 + \overline{\overline{SubF_{\mathcal{B}}}} + \overline{\overline{SubF_{\mathcal{C}}}} \\ |\mathcal{A}| = 3 + |\mathcal{B}| + |\mathcal{C}| \\ |\mathcal{B}| < n \Rightarrow \overline{\overline{SubF_{\mathcal{B}}}} \leq |\mathcal{B}| \\ |\mathcal{C}| < n \Rightarrow \overline{\overline{SubF_{\mathcal{C}}}} \leq |\mathcal{C}| \end{array}\right\} \Rightarrow \overline{\overline{SubF_{\mathcal{A}}}} \leq 1+|\mathcal{B}|+|\mathcal{C}| = |\mathcal{A}|-2 \leq |\mathcal{A}| \Rightarrow \overline{\overline{SubF_{\mathcal{A}}}} \leq |\mathcal{A}|\,.$$

Таким образом, для любой формулы $\mathcal{A}$ Исчисления Высказываний верно, что

$$\overline{\overline{SubF_{\mathcal{A}}}} \leq |\mathcal{A}|\,.$$

◀

1.2 Вывод в Исчислении Высказываний

Прежде чем перейти к понятию вывода в Исчислении Высказываний, необходимо дать определение аксиомы Исчисления Высказываний:

Определение 6. Аксиома Исчисления Высказываний – это любая формула любого из приведённых далее вида, где $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ – произвольные формулы Исчисления Высказываний:

1. $(\mathcal{A} \to (\mathcal{B} \to \mathcal{A}))$,
2. $((\mathcal{A} \to (\mathcal{B} \to \mathcal{C})) \to ((\mathcal{A} \to \mathcal{B}) \to (\mathcal{A} \to \mathcal{C})))$,
3. $((\mathcal{A}\&\mathcal{B}) \to \mathcal{A})$,
4. $((\mathcal{A}\&\mathcal{B}) \to \mathcal{B})$,
5. $((\mathcal{C} \to \mathcal{A}) \to ((\mathcal{C} \to \mathcal{B}) \to (\mathcal{C} \to (\mathcal{A}\&\mathcal{B}))))$,
6. $(\mathcal{A} \to (\mathcal{A} \vee \mathcal{B}))$,
7. $(\mathcal{B} \to (\mathcal{A} \vee \mathcal{B}))$,
8. $((\mathcal{A} \to \mathcal{C}) \to ((\mathcal{B} \to \mathcal{C}) \to ((\mathcal{A} \vee \mathcal{B}) \to \mathcal{C})))$,
9. $((\mathcal{A} \to (\neg\mathcal{B})) \to (\mathcal{B} \to (\neg\mathcal{A})))$,
10. $((\neg\,(\neg\mathcal{A})) \to \mathcal{A})$.

В Исчислении Высказываний используется всего одно правило вывода – **Правило Отделения (*Modus Ponens*)**. Обозначать его будем ***MP***, а запись:

$$\frac{\mathcal{A},\ (\mathcal{A} \to \mathcal{B})}{\mathcal{B}}$$

будет означать, что по правилу ***MP*** из формул $\mathcal{A}$ и $(\mathcal{A} \to \mathcal{B})$ получается формула $\mathcal{B}$.

Определение 7. Выводом в Исчислении Высказываний называется любая конечная последовательность формул Исчисления Высказываний $\mathcal{B}_1,\ \mathcal{B}_2, ..., \mathcal{B}_n$, где для любого i, $(i = 1, ..., n)$

1. либо формула $\mathcal{B}_i$ является аксиомой Исчисления Высказываний,
2. либо найдутся натуральные числа j и k меньшие, чем i, такие, что

$$\frac{\mathcal{B}_j,\ \mathcal{B}_k}{\mathcal{B}_i}.$$

Определение 8. Формула $\mathcal{B}$ называется **выводимой в Исчислении Высказываний**, если существует вывод в Исчислении Высказываний $\mathcal{B}_1,\ \mathcal{B}_2, ..., \mathcal{B}_n$ оканчивающийся формулой $\mathcal{B}$.

2. Алгоритм поиска вывода в Исчислении Высказываний

При попытке разработать алгоритм поиска вывода формулы Исчисления Высказываний, в первую очередь возникает идея: будем генерировать формулы и пытаться из них составить вывод до тех пор, пока не получим вывод заданной формулы. Идея крайне проста, но остается открытым вопрос: *каким образом генерировать формулы*? Сгенерировать все формулы не получится хотя бы потому, что множество пропозициональных переменных бесконечно. Кроме того, стоит цель реализовать алгоритм в виде компьютерной программы, а вычислительные ресурсы ограничены, поэтому процедура генерации формул должна как можно раньше выдавать те формулы, из которых можно составить вывод заданной формул.

2.1 Описание алгоритма

Пусть стоит задача найти вывод выводимой формулы $\mathcal{A}$ Исчисления Высказываний. Во-первых, множество переменных можно ограничить набором переменных, которые входят в формулу $\mathcal{A}$. Во-вторых, заметим, что любой вывод будет содержать аксиомы, поэтому можно генерировать большое количество аксиом, с целью быстрее получить те аксиомы, из которых в дальнейшем получится формула $\mathcal{A}$. Таким образом, предлагается следующий алгоритм:

Алгоритм 1 поиска вывода формулы Исчисления Высказываний

```
1:  for all 𝓑 - подформула формулы 𝓐 do
2:      F ← F ∪ {𝓑}
3:  while вывод не найден do
4:      for all 𝓑 ∈ F do
5:          if формулу 𝓑 можно добавить в конец вывода 𝓓 then
6:              добавить 𝓑 в конец вывода 𝓓
7:              if 𝓑 и 𝓐 равны then
8:                  завершить выполнение алгоритма
9:      F′ ← ∅
10:     for all (𝓑1, 𝓑2, 𝓑3) ∈ F^3 do
11:         for all 𝓑 - формула из списка аксиом do
12:             for all 𝓒 - подформула формулы 𝓑A, B, C [𝓑1, 𝓑2, 𝓑3] do
13:                 F′ ← F′ ∪ {𝓒}
14:     F ← F ∪ F′
```

Операция $\leftarrow$ аналогична операции присвоения в любом языке программирования. По завершении работы алгоритма $\mathcal{D}$ - искомый вывод формулы $\mathcal{A}$.

Важно подчеркнуть, что алгоритм ищет вывод только для **выводимых** формул. Нетрудно заметить, что если на вход этому алгоритму подать формулу, которая не является выводимой, то алгоритм никогда не завершит свою работу.

Аналогичным образом можно искать вывод из множества гипотез: во множество F изначально добавить подформулы каждой из гипотез.

2.2 Оценка количества элементов в множестве F

Через f_n будем обозначать количество элементов в множестве F во время $n-$ой итерации цикла **while** алгоритма 1. До цикла в множестве F столько формул,

сколько подформул в формуле $\mathcal{A}$, а их, по теореме 1, не более чем $|\mathcal{A}|$. Во время $n-$ой итерации цикла **while** в множество F добавляется не более $10 * f_{n-1}^3$ аксиом. Тогда, в первом приближении, будем считать, что

$$f_n \approx 10 * f_{n-1}^3, \text{ где } 1 \leq f_0 \leq |\mathcal{A}| .$$

Данная оценка является **грубой**: никак не учитывается, что в множество F добавляются также все подформулы сгенерированных аксиом. Однако эта оценка, например, позволяет предположить, что попытка реализовать этот алгоритм «в лоб» не завершится успехом, поскольку множество растёт слишком быстро.

3. Программная реализация. Программа «Latium»

Прежде чем перейти непосредственно к реализации алгоритма 1, необходимо решить несколько вспомогательных задач. Их решение и реализация алгоритма образуют программу «Latium», написанную и отлаженную автором данной работы. Если ввести в Яндекс.Переводчик *«логически рассуждающий»*, то на латынь он это переведёт как *«Latium»*, отсюда и произошло название. Программа написана на языке C# версии 7.0 в среде разработки Visual Studio 2017.

В **Приложении А** приведена информация о том, где можно найти исходный код программы.

3.1 Пользовательский интерфейс программы «Latium»

Пользовательский интерфейс программы «Latium» - это простое WPF приложение, внешний вид которого можно увидеть на рисунке 1.

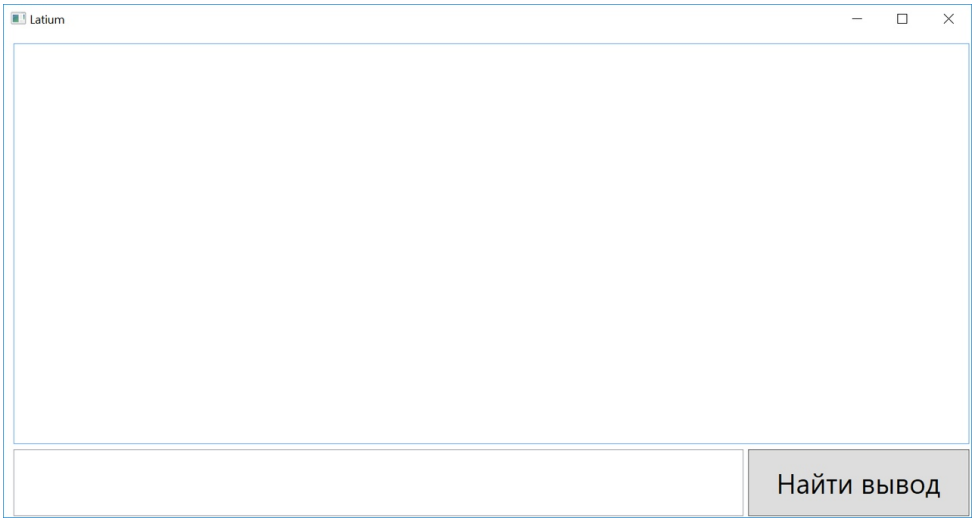

Рис. 1: Интерфейс программы «Latium»

Центральное поле предназначено для вывода на экран результата работы программы - вывода формулы. Поле расположенное внизу - для ввода формулы. Если будет введена не формула, то появится окно, информирующее о том, что введённая строка не является формулой. А по тексту на кнопке ясно, что она запускает выполнение алгоритма поиска вывода.

Отдельно стоит сказать про ввод формул (см. рисунок 2). Для этого был написан класс PropositionalCalculusTextBox, который является наследником класса TextBox. Этот класс имеет ряд отличий от базового класса:

- Если справа от буквы начать вводить число, то цифры образуют нижний индекс этой буквы.
- Если ввести символ $-$, то в поле ввода появится символ $\neg$.
- Если ввести символ $>$, то в поле ввода появится символ $\rightarrow$.
- Если ввести символ $|$, то в поле ввода появится символ $\vee$.

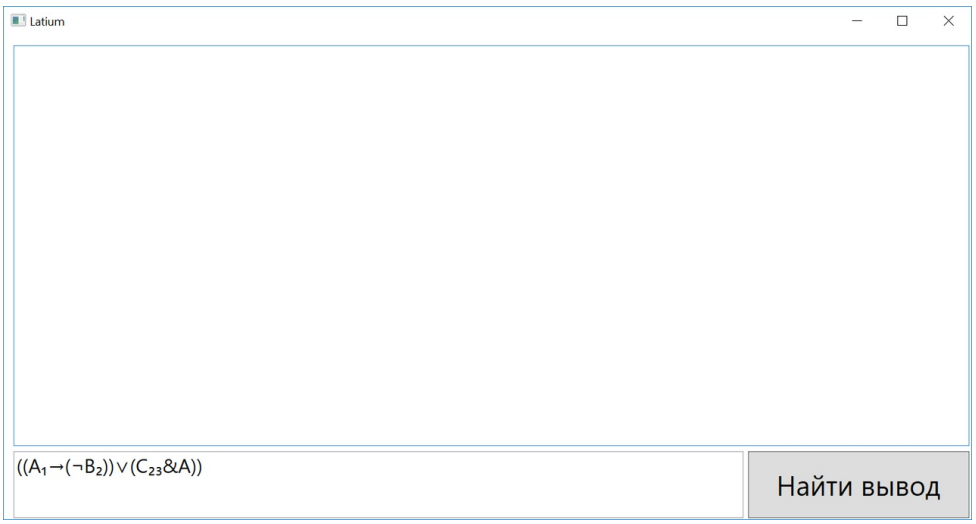

Рис. 2: Демонстрация работы ввода

3.2 Распознавание формулы

В первую очередь, нужно решить задачу распознавания формулы: *является ли данная строка формулой Исчисления Высказываний?*

Для решения этой задачи применялся тот же подход, что используется, например, для решения задачи распознавания арифметических выражений:

1. Разбиваем данную строку на токены, то есть отделяем пропозициональные переменные, пропозициональные связки и технические символы друг от друга.

2. Проверяем расположение пропозициональных связок в строке, например, отрицание должно стоять только перед пропозициональной переменной или левой скобкой.

3. И наконец, по токенам строится, так называемая, обратная польская запись формулы, в ходе построения которой проверяется баланс скобок в строке.

Если все перечисленные шаги выполняются успешно, без ошибок, то данная на вход строка является формулой Исчисления Высказываний, иначе - не является.

В программе распознавание формулы представлено в конструкторе класса FormulaRpn, а методы Tokenize, CheckTokensOrder и TokensToRpn этого класса - это описанные выше пункты соответственно.

3.3 Способ представления формулы

После того, как удалось распознать формулу, естественно возникает другой вопрос: *как представить формулу в памяти компьютера?*

Выше было описано распознавание формул, в ходе которого получалась обратная польская запись этой формулы. Однако можно пойти дальше, а именно, пусть каждая формула в памяти компьютера - это

- либо массив из одного элемента - пропозициональной переменной, если формула является элементарной формулой,

- либо массив из двух элементов - формула $\mathcal{B}$ и символ отрицания, если формула имеет вид $(\neg\mathcal{B})$,

- либо массив из трёх элементов - формула $\mathcal{B}$, формула $\mathcal{C}$ и пропозициональная связка, если формула имеет вид $(\mathcal{B} \vee \mathcal{C})$, или $(\mathcal{B}\&\mathcal{C})$, или $(\mathcal{B} \to \mathcal{C})$.

Такая структура, *во-первых*, позволяет быстро определять какой же вид имеет формула, что полезно, например, при сравнении формул. *Во-вторых*, позволяет быстро получить подформулы, что необходимо при применении правила ***MP***. *В-третьих*, позволяет сократить расходы памяти, например, при выполнении замены переменных в формуле необходимо лишь заменить ссылки на переменные ссылками на формулы, на которые происходит замена. Возможно, достоинства такого представления этим не исчерпываются.

В программе такая форма представления формулы реализована в классах FormulaRpn и Formula, а метод MinimizeRpn класса FormulaRpn по обратной польской записи формулы строит такую структуру.

3.4 Распознавание аксиом

В описании алгоритма 1 есть проверка условия: можно ли добавить формулу в конец вывода. По определению, если формула является аксиомой, то её можно добавить в вывод. Так и возникает задача распознавания аксиом: *является ли данная формула аксиомой Исчисления Высказываний?*

Для решения этой задачи был написан класс FormulaMatcher, метод Match которого умеет сопоставлять формулу с шаблоном, считая переменные в шаблоне некоторыми формулами. Например, первая аксиома Исчисления Высказываний соответствует шаблону $(A \to B)$, но не соответствует шаблону $(A \to A)$.

Таким образом, чтобы узнать является ли данная формула аксиомой, необходимо попытаться эту формулу сопоставить с каждой из формул из определения 6 как с шаблоном. Если формула соответствует одной из этих формул, то она является аксиомой.

3.5 Способ представления вывода

Кроме аксиом, в конец вывода также можно добавить формулу, которая может быть получена по правилу ***MP*** из формул вывода. Метод FindMpPair класса Inference по формуле как раз и пытается найти для формулы её «***MP*** пару». Для того, чтобы ускорить этот поиск, вывод представлен не в виде списка формул, а ввиде словаря с ключами типа формула и со значениями типа целое число - порядковый номер формулы в выводе. Такое представление также позволяет не допустить повторного добавления формулы в вывод, что, очевидно, уменьшает его, а значит поиск по выводу становится быстрее.

Распознавать аксиомы и находить «***MP*** пару» - это необходимые и достаточные методы для реализации процедуры добавления формулы в конец вывода (см. метод Push класса Inference).

3.6 Упрощение вывода

В результате работы алгоритма 1 в вывод добавляется большое количество аксиом, однако не все из них необходимы для вывода итоговой формулы, то есть можно удалить некоторые формулы из полученной последовательности формул, при этом она не перестанет быть выводом. Так и возникла задача упрощения (сокращения) вывода, которую решает алгоритм 2.

Алгоритм 2 сокращения вывода формулы Исчисления Высказываний

1: $\mathcal{D}$ - исходный вывод, $\mathcal{D}'$ - новый пустой вывод
2: $\mathcal{A}$ - последняя формула вывода $\mathcal{D}$
3: AddFormula($\mathcal{A}$)
4: **procedure** AddFormula(формула $\mathcal{A}$)
5: **if** $\mathcal{A}$ - не аксиома **then**
6: $\mathcal{A}$ получена из формул $\mathcal{B}$ и $\mathcal{C}$ по правилу ***MP***,
которые располагаются левее формулы $\mathcal{A}$ в выводе $\mathcal{D}$
7: AddFormula($\mathcal{B}$)
8: AddFormula($\mathcal{C}$)
9: добавить $\mathcal{A}$ в конец вывода $\mathcal{D}'$

По завершении работы алгоритма $\mathcal{D}'$ - сокращенный вывод. Реализован этот алгоритм был как метод Minimize класса Inference.

3.7 Реализаций алгоритма 1

Ранее было выдвинуто предположение, что реализация алгоритма 1 «в лоб» не самая лучшая идея. Так и получилось: программа зависала на самых простых формулах. Для решения этой проблемы необходимо «ускорить» алгоритм.

Заметим, что может возникнуть ситуация, когда необходимые для вывода формулы уже сгенерированы, однако сам процесс генерации ещё не завершён. Поэтому предлагается процессы генерации формул и их добавления в конец вывода выполнять в разных потоках, то есть воспользоваться методами многопоточного программирования языка C#, о которых можно узнать, например, из книги [2]. И замена переменных в аксиомах тоже может происходит в разных потоках. Таким образом, необязательно дожидаться окончания очередной итерации цикла генерации аксиом, а как только вывод будет построен, так сразу завершить работу программы.

Реализация алгоритма 1 с использованием методов многопоточного программирования представлена в классе InferenceFinder.

3.8 Результаты полученные программой «Latium»

Итак, программа написана и отлажена, а значит можно переходить к поиску выводов формул Исчисления Высказываний.

Для начала найдём вывод для формулы $(A_1 \to (B_1 \to A_1))$, которая, очевидно, является аксиомой. Программа определила, что эта формула является аксиомой и вывела вывод состоящий из одной формулы (см. рисунок 3).

Теперь введём формулу $(A \to A)$. В некоторых книгах эта формула является первой формулой, для которой приводится вывод. Программа нашла вывод идентичный выводу этой формулы в книге [1] (см. рисунок 4).

Рассмотрим формулу $((A_1 \vee A_1) \to A_1)$. Оказалось, что программа «Latium» от запуска к запуску может выводить разные выводы, что и произошло в случае рассматриваемой формулы (см рисунки 5, 6 и 7). Это объяснить можно тем, что операционная система во время разных запусков может давать потокам разные интервалы времени, от чего формулы могут генерироваться, а значит и добавляться в вывод, в разном порядке от запуска к запуску. Для усиления этого эффекта формулы при подстановке в аксиомы берутся в случайном порядке (используется стандартный класс Random, генерирующий псевдослучайную последовательность).

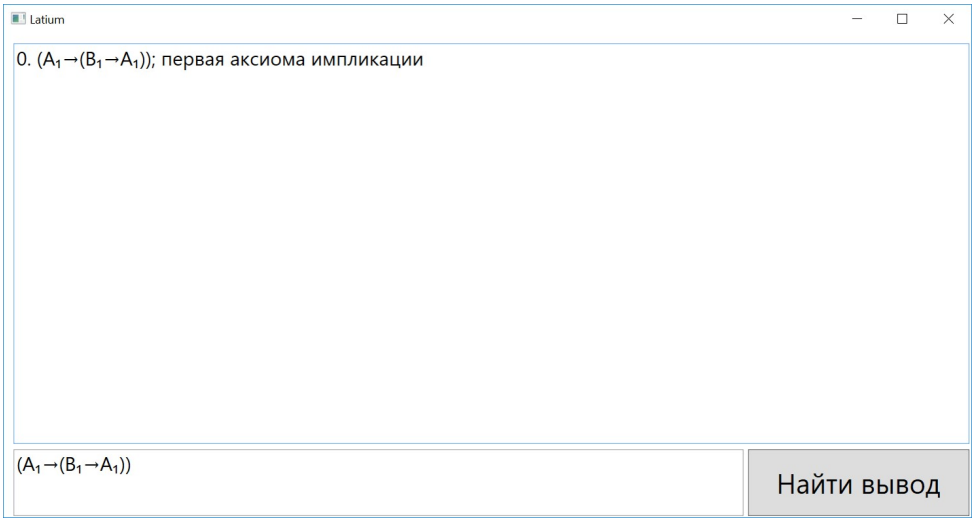

Рис. 3: Вывод формулы $(A_1 \to (B_1 \to A_1))$,

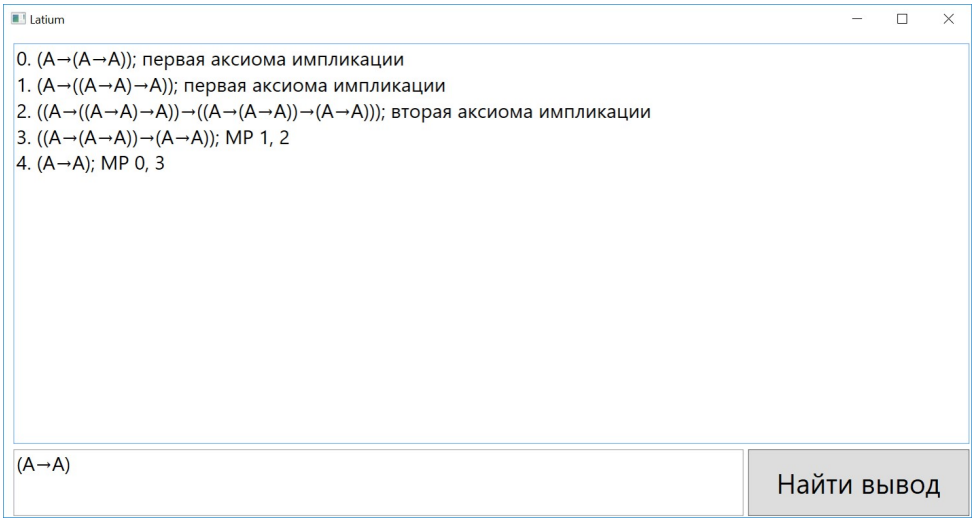

Рис. 4: Вывод формулы $(A \to A)$,

Первый вывод формулы $((A_1 \vee A_1) \to A_1)$(рисунок 5) можно найти в книгах, а вот два оставшихся вывода (рисунки 6 и 7) являются совершенно новыми.

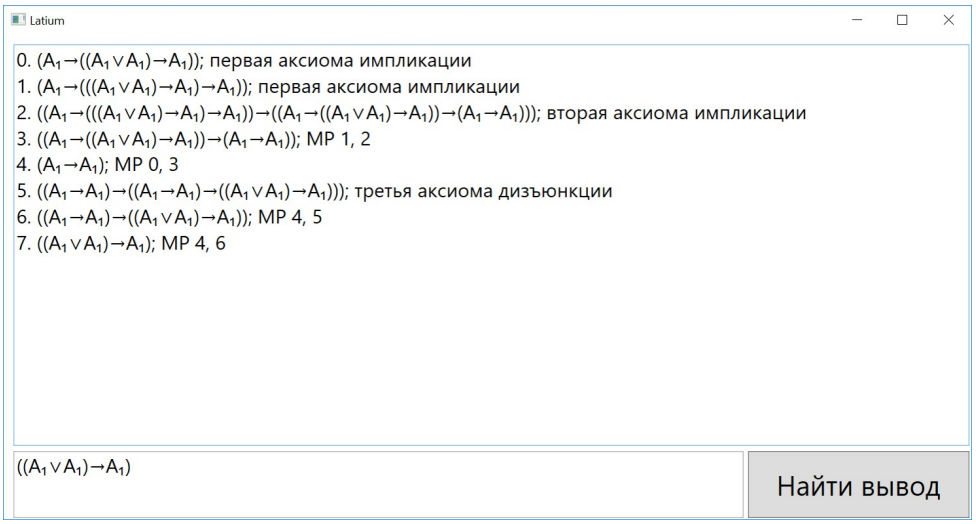

Рис. 5: Первый вывод формулы $((A_1 \vee A_1) \to A_1)$

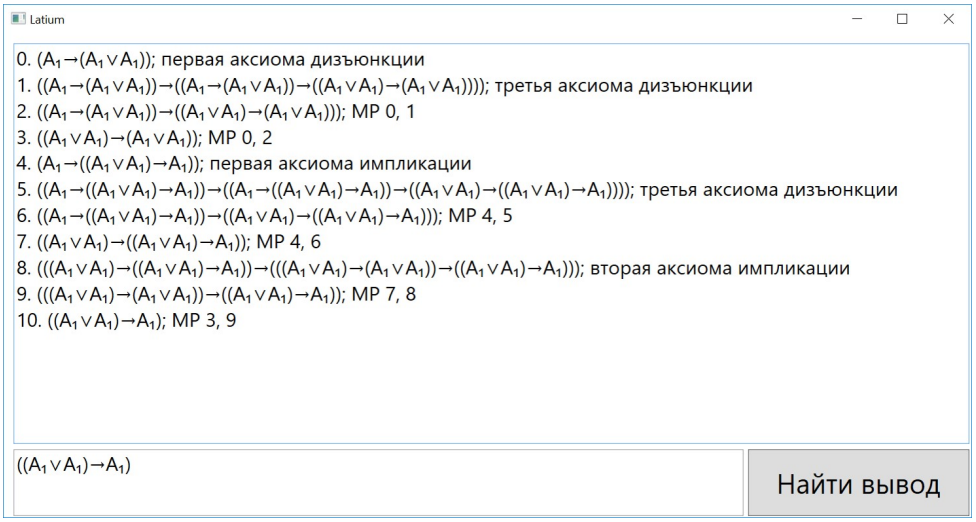

Рис. 6: Второй вывод формулы $((A_1 \vee A_1) \to A_1)$

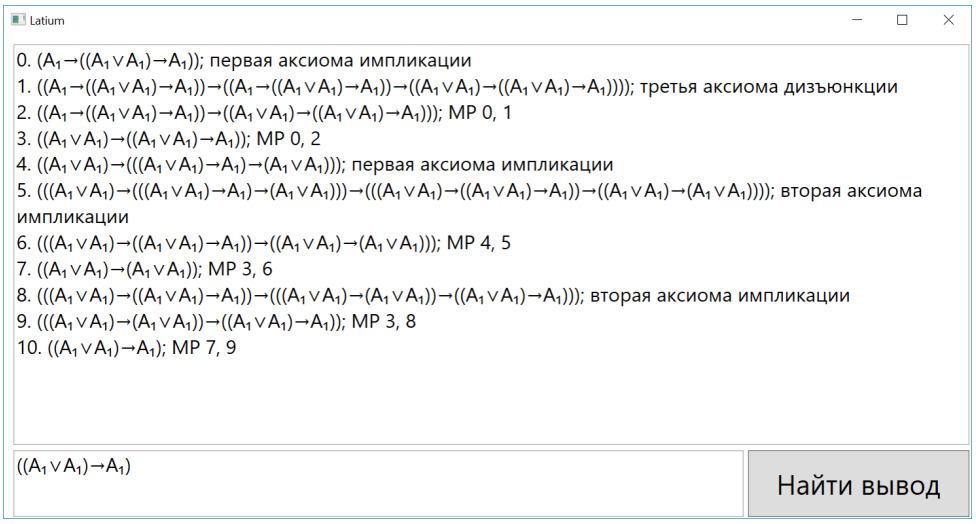

Рис. 7: Третий вывод формулы $((A_1 \vee A_1) \to A_1)$

Заключение

Таким образом цель работы достигнута: написана и отлажена компьютерная программа «Latium», которая ищет вывод введённой с клавиатуры формулы Исчисления Высказываний. В этой программе реализовано распознавание формул Исчисления Высказываний, а для представления формулы в компьютере реализована необычная структура данных, основанная на идее обратной польской записи. Также разработан алгоритм 1, который строит вывод формулы, а также реализован алгоритм 2, который упрощает найденный вывод. Написан модифицированный TextBox, облегчающий ввод формул с клавиатуры. И наконец удалось получить выводы для трёх формул, при этом для формулы $((A_1 \vee A_1) \to A_1)$ были найдены три вывода, два из которых являются совершенно новыми.

В следующем учебном году планируется работа в этом же направлении, а именно решение задач математической логики с применением компьютеров, либо решение задач информатики с применением математической логики. Например, можно продолжить работу в одном из следующих направлений:

- Поиск вывода в Исчислении Предикатов
- Поиск вывода в Исчислении Секвенций
- Изучение и реализация алгоритма Тарского
- Доказательство теорем в формальных системах, например, аксиоматика действительных чисел или аксиоматика арифметики Пеано
- Формальная верификация

Список литературы

[1] Дурнев, В.Г. Элементы теории множеств и математической логики: учеб. пособие / В. Г. Дурнев; Яросл. гос. ун-т. им. П. Г. Демидова. – Ярославль, 2009. 457 с.

[2] Якимова, О. П. Языки программирования. Ч.2: лабораторный практикум / О. П. Якимова, И. М. Якимов, В. Л. Дольников; Яросл. гос. ун-т им. П. Г. Демидова. – Ярославль : ЯрГУ, 2012 – 56 с.

Приложение А

Ссылка на репозиторий с исходным кодом программы «Latium» на GitHub: https://github.com/romarioGI/Latium.

Навигация по репозиторию:

- класс **Inference** (вывод) - Latium/Latium/PropositionalCalculusLibrary/Inference.cs
- класс **Formula** (формула) - Latium/Latium/PropositionLibrary/Formula.cs
- класс **FormulaRpn** (обратная польская запись формулы) - Latium/Latium/PropositionLibrary/FormulaRpn.cs
- класс **FormulaMatcher** («сопоставитель» формул) - Latium/Latium/PropositionLibrary/FormulaMatcher.cs
- класс **PropositionalCalculusTextBox** (ввод формулы) - Latium/Latium/PropositionalCalculusTextBox/PropositionalCalculusTextBox.cs
- класс **InferenceFinder** (реализация алгоритма 1) - Latium/Latium/PropositionalCalculusInferenceFinder/InferenceFinder.cs