СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной практической работе исследуется реализация метода резолюций в логике высказываний на языке программирования C++. Метод резолюций является фундаментальным инструментом в области автоматического доказательства теорем в логике высказываний.

Актуальность данной работы обуславливается значимостью метода резолюций в области формальных методов и искусственного интеллекта. Применение данного метода позволяет автоматизировать процесс вывода логических заключений, что важно для разработки и верификации программного обеспечения, а также для исследований в области математики и логики.

Целью данной практической работы является изучение теоретических основ метода резолюций, его реализация на языке C++ с помощью фреймворка Qt и анализ результатов работы программы.

Для достижения поставленной цели были сформулированы следующие задачи:

1. Изучить основные принципы метода резолюций в контексте логики высказываний.
2. Разработать программу на языке C++, используя фреймфорк Qt, реализующую метод резолюций.
3. Провести тестирование программы на различных логических задачах и проверить её корректность.
4. На основании проделанной работы сделать вывод о перспективах использования метода резолюций в современных исследованиях и практических приложениях.

ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

§ 1.1. Введение

В данной главе мы рассмотрим теоретичские основы метода резолюций. В начале будет приведена историческая справка о развитии логики и, в частности, математической логики. Далее будут введены основные понятия метода для логики высказываний и рассмотрен метод резолюций для логики высказываний. Также будет рассмотрена теорема о полноте для логики высказываний, а в конце главы обсуждаются стратегии метода, применения метода для доказательства теорем и решения задачи планирования действий, взаимосвязь метода и логического программирования.

§ 1.2. Историческая справка

Термином **«логика»** называется наука, изучающая формы и законы мышления, способы построения доказательств и опровержений различных утверждений. [1] Логика берет начало от работ древнегреческого философа Аристотеля (4 век до нашей эры). Он первым обратил внимание на то, что при выводе одних утверждений из других исходят не из конкретного содержания рассуждений, а из взаимоотношения между их формами. Логика Аристотеля усовершенствовалась на протяжении многих веков. Значительный качественный прогресс в развитии логики наступил с применением в логике математических методов.

Возникновение науки, которая называется **математической логикой**, связывают с работами аглийского математика и логика Д. Буля. Им была создана **алгебра логики** – результат применения к логике алгебраических методов. В 20 веке на базе математической логики была разработана теория алгоритмов. В разработку этой теории внесли существенный вклад английский математик А. Тьюринг и американский математик Э. Пост.

Математическая логика в течение всего периода развития имела применение как в математике, так и вне ее. Весьма значительны применения математической логики в кибернетике и информатике. Так, одной из основных задач искусственного интелекта является разработка моделей представления знаний. Кроме стредств описания знаний, модель должна обладать и дедуктивными возможностями - уметь получать следствия из некторой исходной информации. Этим требованиям в полной мере удовлетворяют логические модели, в основе которых лежит математическая логика.

§ 1.3. Основные понятия логики высказываний

Логика высказываний - это простейший раздел математической логики, изучающий сложные высказывания, образованные из простых, и их взаимоотношения.

Основным понятием логики высказываний явзяется **высказывание** - утвердительное предложение, которое может быть истинным или ложным, но не тем и другим вместе. [2] Примеры высказываний: «Снег белый» , «Яблоко сладкое», «Яблоко большое». Каждое высказывание имеет «значение истинности». Условимся использовать заглавные буквы для обозначения высказываний. Например обозначим данные высказывания:

- A = «Снег белый»
- B = «Яблоко сладкое»
- C = «Яблоко большое»

Символы A, B, C и т.д., которые используются для обозначения высказываний, называются атомарной формулой или **литералом**.

Из высказываний мы можем строить составные высказывания, используя **логические связки**. В логике высказываний используются пять логических связок:

1. $\vee$ - дизъюнкция (или)
2. $\wedge$ - конъюнкция (и)
3. $\neg$ - отрицание (не)
4. $\rightarrow$ - импликация (если ... , то ...)
5. $\leftrightarrow$ - равносильность (тогда и только тогда, когда)

Составим таблицу истинности связок:

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$A \leftrightarrow B$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

Таблица 1. Таблица истинности связок

Эти пять связок используются для построения более сложных составных высказываний. Пример составного высказывания: B $\wedge$ C = «Яблоко сладкое и Яблоко большое». В логике высказываний высказывание или составное высказывание называется **формулой**. Формулы в логике высказываний определяются рекурсивно следующим образом:

1. Атом (литерал) есть формула.
2. Если A - формула, то $\neg$A - формула.
3. Если A и B - формулы, то (A $\wedge$ B), (A $\vee$ B), (A $\rightarrow$ B), (A $\leftrightarrow$ B) - формулы.
4. Никаких формул, кроме порожденных применением указанных выше правил, нет.

Существую формулы, которые истинны при всех возможных интерпретациях и ложны при всех возможных интерпретациях, такие формулы называют общезначимыми и противоречивыми соответственно.

Говорят, что две формулы A и B эквивалентны, тогда и только тогда, когда значения истинности A и B совпадают при каждой интерпретации A и B.

Одна из основных целей изучения логики состоит в получении формального аппарата для доказательства того, является ли данное утверждение следствием других. Это приводит нас к понятию **«логического следствия»**.

Пусть даны формулы $F_1, F_2, ..., F_n$ и формула G. Говорят, что G есть **логическое следствие** формул $F_1, F_2, ..., F_n$ (или G логически следует из $F_1, F_2, ..., F_n$) если для всякой интерпретации ϕ из того, что $\phi(F_1) = \phi(F_2) = ... = \phi(F_n) = 1$, следует, что $\phi(G) = 1$. $F_1, F_2, ..., F_n$ называются аксиомами (посылками) G.

Понятие логического следствия тесно связано с понятие выполнимости. Множество формул $\{F_1, F_2, ..., F_n\}$ называется выполнимым, если существует такая интерпретация ϕ, что $\phi(F_1) = \phi(F_2) = ... = \phi(F_n) = 1$.

§ 1.4. Метод резолюций в логике высказываний

Для изучения метода резолюций нам понадобится следующее утверждение.

Теорема 1. Формула G является логическим следствием формул $F_1, F_2, ..., F_n$ тогда и только тогда, когда множество формул $L = \{F_1, F_2, ..., F_n, \neg G\}$ невыполнимо.

Доказательство: Пусть формула G является логическим следствием формул $F_1, ..., F_k$. Предположим, что множество L выполнимо. Это означает, что существует интерпретация ψ такая, что $\psi(F_1) = ... = \psi(F_k) = \psi(\neg G) = 1$. Но если $\psi(F_1) = ... = \psi(F_k) = 1$, то $\psi(G) = 1$, поскольку G - логическое следствие формул $F_1, ..., F_k$. Полученное противоречие $\psi(G) = 1$ и $\psi(\neg G) = 1$ доказывает, что множество формул $\{F_1, ..., F_k, \neg G\}$ невыполнимо.

Пусть теперь множество L невыполнимо. Рассмотрим интерпретацию ϕ такую, что $\phi(F_1) = ... = \phi(F_k) = 1$. Поскольку L невыполнимо, имеем $\phi(\neg G) = 0$. Если $\phi(\neg G) = 0$, то $\phi(G) = 1$. Следовательно из равенств $\phi(F_1) = ... = \phi(F_k) = 1$ следует равнство $\phi(G) = 1$. Это означает, что G - логическое следствие множества формул $F_1, ..., F_k$.

Метод резолюций является методом доказательства того, что формула G является логическим следствием формул $F_1, F_2, ..., F_k$. Исходя из **Теоремы 1** отметим, что задача о логическом следствии сводится к задаче о выполнимости. Действиетельно, формула G есть логическое следствие формул $F_1, F_2, ..., F_k$ тогда и только тогда, когда множество формул $\{F_1, F_2, ..., F_k, \neg G\}$ невыполнимо. Таким образом, первая особенность метода резолюций заключается в том, что он устонавливает невыполнимость

множества формул. Вторая особенность метода - он оперирует не с произвольными формулами, а с **дизъюнктами**(элементарными дизъюнкциями литералов). Для удобства, на дизъюнкт мы будем смотреть, как на множество литералов, то есть не будем различать дизъюнкты, которые получаются один из другого с помощью коммутативности и ассоциативности дизъюнкции, а также идемпотентности. Также пусто дизъюнкт, не содержащий литералов, принято обозначать символом $\square$. Такой дизъюнкт ложен при любой интерпретации.

Метод резолюций в логике высказываний основан на **правиле резолюций**. Правилом резолюций в логике высказываний называется следующее правило: из дизъюнктов $A \vee B$ и $\neg A \vee C$ выводим дизъюнкт $B \vee C$.

Например, из дизъюнктов $\neg X \vee Y \vee Z$ и $X \vee \neg Y$ вывродим дизъюнкт $Y \vee Z \vee \neg Y$. Обратим внимание на то, что в первых двух дизънктах есть еще одна пара противоположных литералов. Условимся, что можно применять правило резолюций не обязательно к самым левым литералам. Пусть S - множество дизънктов. Выводом из S называется последовательность дизъюнктов $D_1, D_2, ..., D_n$ такая, что каждый дизъюнкт этой последовательности принадлежит S или следует из предыдущих по правилу резолюций.

Применение метода резолюций основано на следующем утверждении, которое называется теоремой о полноте метода резолюций.

Теорема 2. О полноте метода резолюций. Множество дизъюнктов логики высказываний S невыполнимо тогда и только тогда, когда из S выводим пустой дизъюнкт $\square$.

Доказательство: Отметим, что правило резолюций сохраняет истинность. Это означает, что если $\phi(\neg X \vee F) = 1$ и $\phi(X \vee G) = 1$ для некоторой интерпретации ϕ, то $\phi(F \vee G) = 1$.

Докажем в начале достаточность.

Пусть из S выводим пустой дизъюнкт. Предположим противное: множество S выполнимо, т.е. существует интерпретация ψ, при которой все дизъюнкты из S истинны. Выводимость пустого дизъюнкта из S означает, что существует последова- тельность дизъюнктов $D_1, ..., D_n = \square$, каждый дизъюнкт которой принадлежит S или получается из предыдущих по правилу резолюций. Если дизъюнкт D_j из этой последовательности принадлежит S, то по предположению $\psi(D_j) = 1$. Если же он получается из предыдущих по правилу резолюций, то также $\psi(D_j) = 1$, поскольку правило резолюций сохраняет истинность. При $i = n$ получаем, что $\psi(\square) = 1$. Противоречие показывает, что предположение о выполнимости множества S – ложное предположение. Следовательно, S невыполнимо. Достаточность доказана.

Докажем необходимость. Доказательство проведем индукцией по следующему параметру $d(s)$: это сумма числа вхождений литералов в дизъюнкты из S минус число дизъюнктов.

Пусть множество дизъюнктов S невыполнимо. Если пустой дизъюнкт принадлежит S, то он выводим из S (вывод в этом случае состоит из одного пустого дизъюнкта) и необходимость теоремы доказана. Будем считать в силу этого, что $\square \notin S$. При этом предположении каждый дизъюнкт содержит хотя бы один литерал и поэтому $d(S) \geq 1$.

База индукции: $d(S) \geq 1$. Если $d(S) = 1$, то все дизъюнкты состоят из одного литерала. Поскольку множество S невыполнимо, то в нем должна найтись пара противоположных литералов X и $\neg X$. В таком случае пустой дизъюнкт выводим из S, соответствующий вывод содержит три дизъюнкта: $X, \neg X, \square$.

Шаг индукции: $d(S) > 1$. Предположим, что для любого множества дизъюнктов такого, что $d() < d(S)$ необходимость теоремы доказана. Пусть $S = \{D_1, D_2, ..., D_{k-1}, D_k\}$.

Так как $d(S) > 1$, то в S существует хотя бы один неодноэлементный дизъюнкт. Будем считать, что это дизъюнкт D_k , т.е. $D_k = L \vee D'_k$, где L – литерал и $D_k \neq \square$. Наряду с множеством дизъюнктов S рассмотрим еще два множества дизъюнктов:

$$S_1 = \{D_1, D_2, ..., D_{k-1}, L\}$$

$$S_2 = \{D_1, D_2, ..., D_{k-1}, D'_k\}$$

Ясно, что S_1 и S_2 невыполнимы и что $d(S_1) < d(S)$ и $d(S_2) < d(S)$. По предположению индукции из S_1 и S_2 выводим пустой дизъюнкт. Пусть $A_1, A_2, ..., A_i, ..., A_{l-1}, A_l = \square$ - вывод пустого дизъюнкта из S_1 и $B_1, B_2, ..., B_j, ..., B_{m-1}, B_m = \square$ - вывод пустого дизъюнкта из S_2. Если в первом выводе не содержится дизъюнкта L, то эта последовательность дизъюнктов будет выводом из S и необходимость теоремы доказана. Будем считать, что L содержится в первом выводе, пусть $A_i = L$. Аналогично предполагаем, что $B_j = D'_k$.

Если дизъюнкт E получается из дизъюнктов E_1 и E_2 по правилу резолюций, то будем говорить, что E непосредственно зависит от E_1 и от E_2. Транзитивное замыкание отношения непосредственной зависимости назовем отношением зависимости.(Другими словами, E зависит от E' , если существуют дизъюнкты $E_1, \ldots, E_n$ такие, что $E = E_1, E_n = E'$ и E_1 непосредственно зависит от E_2 , E_2 непосредственно зависит от $E_3, \ldots, E_{n-1}$ непосредственно зависит от E_n). Преобразуем второй вывод следующим образом: к дизъюнкту B_j и всем дизъюнктам, которые от него зависят, добавим литерал L. Новая последовательность (1) дизъюнктов $B_1, B_2, ..., B'_j = D'_k \vee L, B'_{j+1}, ..., B'_m$ будет выводом из S. Если дизъюнкт B_m не зависит от B_j , то $B'_m = \square$. Это означает, что из S выводим пустой дизъюнкт, что и требовалось доказать. Предположим, что B_m зависит от B_j . Тогда $B'_m = L$. Преобразуем теперь первый вывод: на место дизъюнкта A_i (равного L) в этой последовательности подставим последовательность (1). Получим последовательность $A_1, ..., A_{i-1}, B_1, ..., B'_j, B'_{j+1}, ..., B'_m = L, A_{i+1}, ..., A_l = \square$. Эта последовательность является выводом пустого дизъюнкта из множества дизъюнктов S. Следовательно, если множество S невыполнимо, то из S выводим пустой дизъюнкт $\square$.

§ 1.5. Применение метода резолюций в логике высказываний

Для доказательства того, что формула G является логическим следствием множества формул $F_1, ..., F_k$, метод резолюций применяется следующим образом.

Сначала составляется множество формул $T = \{F_1, \ldots, F_k, \neg G\}$. Затем каждая из этих формул приводится к конъюнктивной нормальной форме и в полученных формулах зачеркиваются знаки конъюнкции. Получается множество дизъюнктов S. И, наконец, ищется вывод пустого дизъюнкта из S. Если пустой дизъюнкт выводим из S, то формула G является логическим следствием формул $F_1, ..., F_k$. Если из S нельзя вывести $\square$, то G не является логическим следствием формул $F_1, ..., F_k$.

Приведем **пример** использования метода резолюций. Покажем, что формула $G = Z$ является логическим следствием формул $F_1 = \neg X \vee Y \to X \wedge Z, F_2 = \neg Y \to Z$. Сформируем множество формул $T = \{F_1, F_2, \neg G\}$. Приведем формулы F_1 и F_2 к КНФ(фомрула $\neg G$ уже находится в КНФ). Получим: $F_1 = X \wedge (\neg Y \vee Z), F_2 = Y \vee Z$.

Тогда множество дизъюнктов $S = \{X, \neg Y \vee Z, Y \vee Z, \neg Z\}$. Из множества S легко выводится пустой дизъюнкт: $\neg Y \vee Z, \neg Z, \neg Y, Y \vee Z, Y, \square$.

Следовательно, формула G является логическим следствием формул F_1, F_2.

ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

§ 2.1. Введение

Для реализации метода резолюций в логике высказыаний была написана программа на языке программирования C++ с использованием фреймворка Qt. Для данной программы были поставлены следующие функциональные требования:

1. Ввод данных:
   - Программа должна позволять пользователю вводить логические выражения в корректном формате.
2. Обработка данных:
   - Реализация метода резолюций для логики высказываний.
   - Преобразование входных логических выражений в конъюнктивную нормальную форму.
   - Генерация новых дизъюнктов посредством правила резолюций до получения пустого дизъюнкта или до невозможности дальнейшего разрешения.
3. Вывод данных:
   - Отображение результатов выполнения метода резолюций.
   - Отображения множества дизъюнктов S и вывода из него.
   - Сообщение пользователю о том, является ли введенная теорема логическим следствием введенных формул.
4. Пользовательский интерфейс:
   - Предоставление графического интерфейса для ввода данных и отображения результатов.
   - Кнопки для запуска процесса резолюции и удаления введенных данных.

Чтобы знаки логических операций отображались но всех устройствах одинаково и без ошибок, в рамках данного приложения для логических операций введены следующие эквивалентные символы:

- Операция $\neg$ обозначается символом «!».
- Операция $\vee$ обозначается символом «+».
- Операция $\wedge$ обозначается символом «*».

- Операция $\rightarrow$ обозначается символом «->».
- Операция $\leftrightarrow$ обозначается символом «==».

§ 2.2. Реализация

Программа состоит из трех основных классов:

- **MainWindow**
- **Formula**
- **Resolution**

Каждый класс выполняет определенную роль в реализации функционала приложения. На рисунке 1 показана диграмма классов[6]:

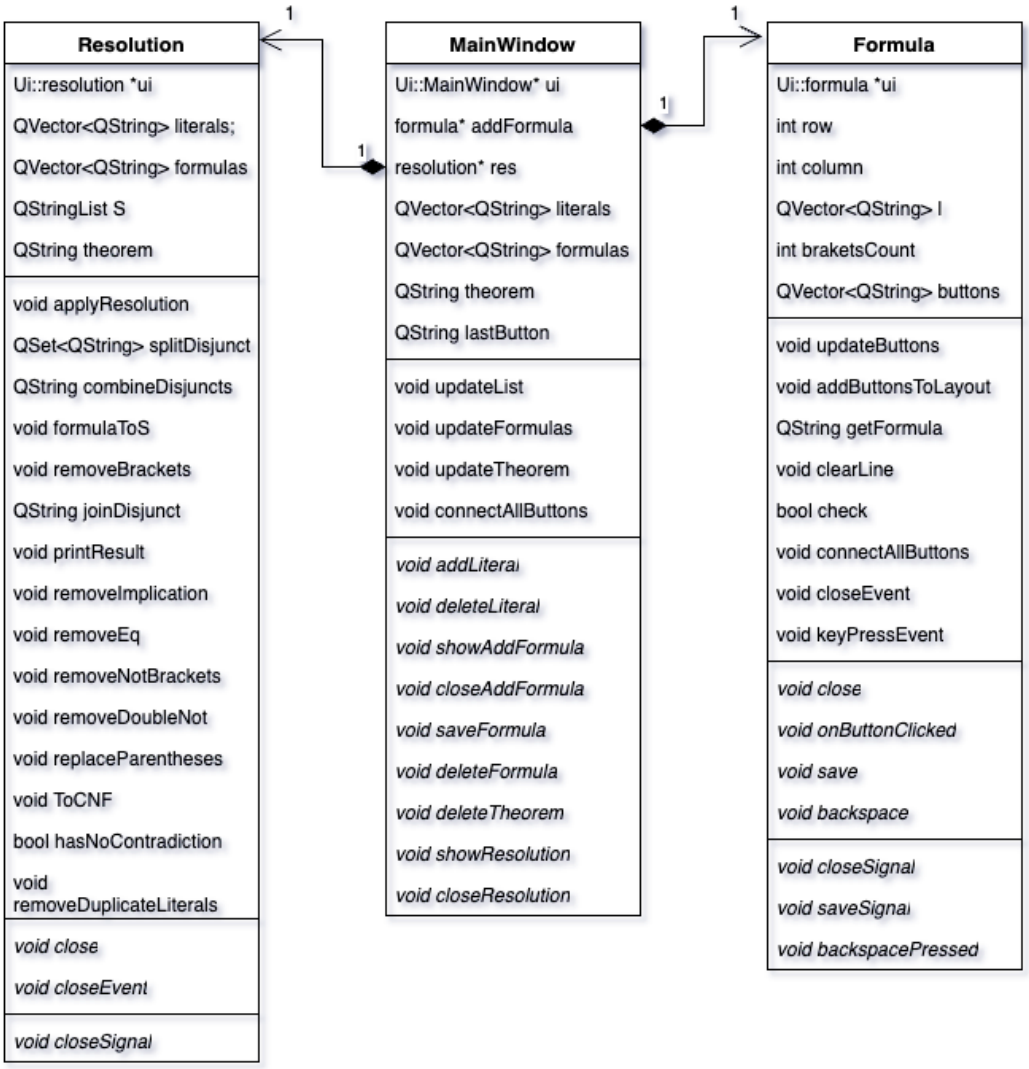

Рис. 1. Диаграмма классов

1. **MainWindow**

   Данный класс отвечает за основное окно приложения. Он предоставляет пользователю интерфейс для добавления литералов и управления диалоговыми окнами для ввода формул и проверки теоремы методом резолюций. Так же данный класс обрабатывает пользовательские события, такие как нажатие кнопок, ввод формулы и запуск процесса резолюции.

2. **Formula**

   Класс Formula отвечает за ввод логических функций. Он предоставляет диалоговое окно, в котором пользователь может ввести логическое выражение. Также важнейшей функцией этого класса является проверка на корректность введенной формулы. Введенная формула передается в основной класс для дальнейшей обработки.

3. **Resolution**

   Данный класс реализует метод резолюций для проверки теоремы. Он отвечает за выполнение следующих действий:

   - Приведение каждой формулы из множества допущений к КНФ.
   - Строится новая формула - отрицание введенной теоремы, которая так же приводится к КНФ.
   - Из всех полученных КНФ удалением знака «*» составляется множество дизъюнктов S.
   - Ищется вывод из множества S.
   - Отображение результатов выполнения алгоритма пользователю.

Классы взаимодействуют между собой следующим образом: при запуске программы создается объект класса MainWindow, отображающий основное окно. При нажатии кнопки «Задать гипотезу» или «Добавить допущение» вызывается диалоговое окно Formula для ввода формулы. При нажатии кнопки «Сохранить» в этом диалоговом окне введенная формула проверяется на корректность и сохраняется в классе MainWindow. Окно Resolution с результатами метода резолюций вызывается при нажатии кнопки «Проверить гипотезу» на основном окне приложения.

Рассмотрим структуру каждого класса отдельно.

§ 2.3. Класс MainWindow

На рисунке 2 изображен интерфейс основного окна.

Рис. 2. Основное окно приложения

Рассмотрим основные поля класса:

- `QVector<QString> literals` - поле содержащее массив литералов (латинские буквы в верхнем регистре «A-Z»).
- `QVector<QString> formulas` - массив введенных формул (допущений).
- `QString theorem` - введенная теорема.
- `QString lastButton` - вспомогательная переменная, содержащая название последней нажатой кнопки. Необходима, для корректного использования одного класса Formula для сохранения формул отдельно от теоремы.

- `formula* addFormula` - диалоговое окно добавления формулы.
- `resolution* res` - диалоговое окно для применения метода резолюций и отображения результата.
- `int count` - количество добавленных литералов.

Так как взаимодействие пользователя с программой реализовано посредством нажатия на кнопки, необходимо соединить событие нажатия определенной кнопки с соответствующим слотом. Для этого была написана функция `void connectAllButtons()`.

Для добавления литерала написан слот `void addLiteral()` который добавляет следующую букву латинского алфавита в список литералов. Аналогично для удаления последнего добавленного литерала написан слот `void deleteLiteral()`.

Для вызова дилогового окна добавления формулы написан метод `showAddFormula`. Данное окно может быть закрыто с сохранением введенной формулы (если она корректна) при помощи слота `void saveFormula()` или без сохранения при помощи метода `void closeAddFormula()`. Удаление последней добавленной формулы реализовано слотом `void deleteFormula()`, а удаление теоремы слотом `void deleteFormula()`. Все изменения отображаются на главном окне с помощью методов `void updateList()`, `void updateFormulas()`, `void updateTheorem()` для обновления списка переменных, списка формул и теоремы соответственно.

Для получения результата метода резолюций вызывается метод `void showResolution()`. Диалоговое окно с результатами закрывается слотом `void closeResolution()`.

В рамках данного отчета рассматривать подробно реализации всех функций данного класса не будем, так как они обеспечивают пользовательский интерфейс и по сути не имеют отношения к методу резолюций.

§ 2.4. Класс Formula

На рисунке 3 изображено диалоговое окно для ввода формулы. Рассмотрим ос-

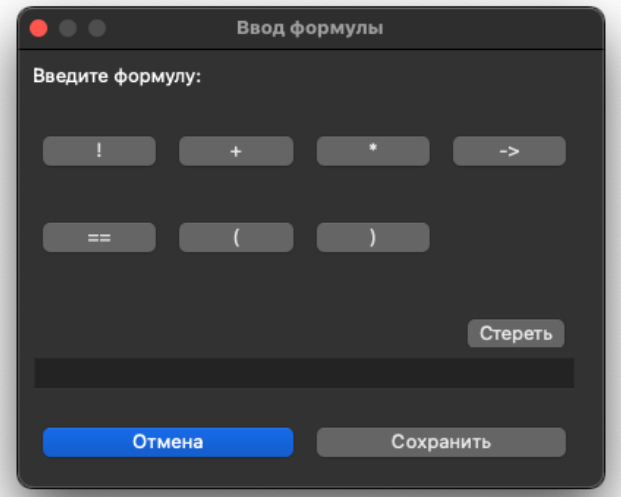

Рис. 3. Диалоговое окно ввода формулы

новные поля класса:

- `int row, column`. Кнопки для ввода символов расположены в сетку по 4 кнопки в ряд. Данные переменные нужны для корректоного отображения кнопок.
- `QVector<QString> buttons` - массив названий кнопок операций.
- `QVector<QString> l` - массив названий кнопок литералов. Он заполняется массивом литералов из класса MainWindow при вызове диалогового окна.
- `int braketsCount` - счетчик введенных скобок («(» и «)»). Нужен для проверки корректности введенной формулы.

Рассмотрим основные методы и слоты этого класса. `void clearLine()` - метод для очистки введенной строки. Вызывается при отображении диалогового окна.

`QString getFormula()` - геттер для введенной формулы. С его помощью формула добавляется в массив формул в классе MainWindow.

При добавлении литералов, необходимо обновить кнопки для ввода. Для этого написан метод `void updateButtons()`, а метод `void connectAllButtons()` соединяет все кнопки с соответствующими слотами.

Проверка корректности ввода формулы осуществляется следующим образом. Для этого написан метод `bool check()` который возвращает значение `true` только, если в введнной строке содержиться хотя бы 1 литерал, если нет двух литералов подряд

без операции между ними и количество открытых скобок равно количеству закрытых. Также, так как есть возможность удалять введенный символ при помощи слота `void backspace()`, соответственно возможно стереть часть операции «==» или «->». Функция также проверяет не стерта ли часть этих операций. Для ограничения ввода написан соответствующий слот `void onButtonClicked(const QString & text)`, который принимает текст нажатой кнопки и проверяет не введена ли вторая операция подряд.

Для сохранения фомрмулы предназначен слот `void save()`, который посылает сигнал сохранения `void saveSignal()` и закрывает диалоговое окно. Аналогично для закрытия окна без сохранения реализован слот `void close()`.

Аналогично классу MainWindow класс Formula не имеет отношения к реализации метода резолюций, поэтому не будем рассматривать подробно реализацию каждого метода.

§ 2.5. Класс Resolution

Данный класс отвечает за приминение метода резолюций и вывод результата. На рисунке 4 изображен интерфейс диалогового окна.

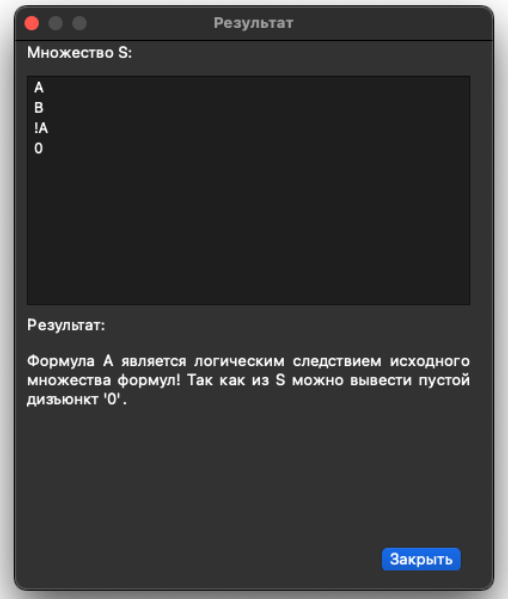

Рис. 4. Диалоговое окно ввода формулы

Рассмотрим поля класса Resolution:

- `QVector<QString> literals` - массив литералов, который заполняется массивом из основного класса MainWindow.

- `QVector<QString> formulas` - аналогичный массив формул (допущений).
- `QStringList S` - массив дизъюнктов.
- `QString theorem` - введенная теорема.

Приминение метода резолюций можно разделить на следующие этапы:

1. Приведение формул к КНФ.
2. Отрицание теоремы и приведение полученной формулы к КНФ.
3. Составление множества дизъюнктов S.
4. Нахождение вывода из множества S при помощи правила резолюций.

При реализации программы одним из самых сложных этапов стало приведение произвольной формулы к КНФ. Алгоритм приведения формулы к КНФ[3]:

1. Избавиться от операций эквивалетности «==» и импликации «->», заменив их конъюнкцией, дизъюнкцией и отрицанием. Для этого надо использовать следующие формулы: $A \to B = \neg A \vee B$ и $A \leftrightarrow B = (\neg A \vee B) \wedge (A \vee \neg B)$.
2. Заменить знаки отрицания, относящиеся к выражению, знакакми отрицания относящимися к литералам. Для этого надо использовать правила Де Моргана: $\neg(A \vee B) = \neg A \wedge \neg B$ и $\neg(A \wedge B) = \neg A \vee \neg B$
3. Избавиться от двойного отрицания.
4. Применить, если нужно, к операциям конъюнкции и дизъюнкции свойства дистрибутивности.

Отметим, что для метода резолюций не принципиальна минимальность КНФ, поэтому к полученному выражению можно не применять правила поглащения.

Для избавления от операции эквивалентности «==» был написан метод `void removeEq(QString& str)`. Данный метод применятется пока в строке с формулой содержится «==». Он определяет операнды справа и слева от знака операции (это может быть просто литерал, отрицание литерала, скобка или отрицание скобки)[5]. Затем составляется новая строка `newStr` с выражением с использованием этих операндов, часть старой формулы меняется на `newStr`.

Аналогичным способом реализован метод для избавления от импликации `void removeImplication(QString& str)`. Он также определяет левый и правый опренд и заменяет импликацию эквивалентным выражением.

Для избавления от отрицания выражений написан метод `void removeNotBrackets (QString& str)`. Он работает следующим образом. Пока в строке формулы содержится

«!(» (отрицание выражения) перебираем символы строки, если этот символ - отрицание, то возможны два варианта: это либо отрицание литерала, тогда ничего делать не надо, или это отрицание выражения. В таком случае определяется подстрока, заключенная между скобками. Если эта подстрока содержит операции, то применяются правила Де Моргана, если же операций нет, то это лишние скобки и их можно просто удалить.

Двойное отрицание удаляется с помощью метода `void removeDoubleNot(QString& str)` он использует регулярное выражение и заменяет все вхождения «!!» на пустую строку.

Последний шаг в приведении к КНФ - это раскрытие скобок и применение закона дистрибутивности для конъюнкции. Данные действия выполняет метод `void replaceParentheses (QString& str)`. Он пока строка изменяется раскрывает скобки следую следующим правилам: если в скобках содержится только умножение, то скобки лишние и их можно удалить, если в скобках только плюсы, то скобки не удаляются, так как подстрока содержащаяся в этих скобках является дизъюнктом. Если же внутри скобок есть как сложение, так и умножение, то применяется закон дистрибутивности и эти скобки заменяются на произведение скобок, содержащих только сложение (что и является КНФ).

Таким образом описанные выше методы приводят строку к КНФ. Для более удобного приведения написана функция `void ToCNF(QString& str)`.

```
1   void resolution::ToCNF(QString &str)
2   {
3     removeEq(str);
4     removeImplication(str);
5     removeNotBrackets(str);
6     removeDoubleNot(str);
7     replaceParentheses(str);
8     QStringList res;
9     QStringList tmp = str.split('*');
10    for (QString& s : tmp) {
11      removeBrackets(s);
12      if(hasNoContradiction(s)){
13        removeDuplicateLiterals(s);
14        res.append(s);
15      }
16    }
17    str = res.join('*');
18  }
```

Листинг 1. Метод void ToCNF(QString& str).

После вызова описанных выше методов получаем формулу в КНФ, но ее слагаемые могут иметь лишние скобки, поэтому просто можно удалить все скобки методом `void removeNotBrackets(QString& str)`. Также строка разбивается на подстроки - дизъюнкты, и с помощью метода `bool hasNoContradiction(const QString &formula)` происходит проверка, содержит ли дизъюнкт литерал и его отрицание. Если содержит, то этот множитель можно удалить так как он тождественно равен единице.

В дальнейшем для реализации правила резолюций нам понадобится удалять по-

вторяющиеся литералы с дизъюнкте. Для этого написан метод `void removeDuplicateLiterals (QString &formula)`.

После приведения формул к КНФ, необходимо составить множество дизъюнктов S. Для этого написан метод `void formulaToS (const QString& str)` который принимает в качестве параметра строку с формулой в КНФ и разделяет ее по символу «*». Полученные дизъюнкты добавляются в массив дизъюнктов S.

Только теперь, когда составлено множество исходных дизъюнктов S. Можно приступить к применению правила резолюций для вывода новых дизъюнктов. Эту функцию выполняет метод `void applyResolution()`. Он работает по следующему принципу: пока есть новый добавленный дизъюнкт перебираются все возможные пары дизъюнктов (`disjunct1` и `disjunct1`) из множества S и, если в них есть противоположные слагаемые, то создается новый массив слагаемых `QSet<QString> resolven` в него записываются все слагаемые из обоих дизъюнктов, кроме противоположных. Далее, если получилась пустая резольвента, то в S добавляется пустой дизъюнкт, если же резольвента не пустая и не содержит противоположных переменных (иначе она равна единице) то она добавляется в множество S.

```
1   void resolution::applyResolution()
2   {
3     bool newDisjunctAdded = true;
4     while (newDisjunctAdded) {
5       newDisjunctAdded = false;
6       QStringList newDisjuncts;
7       for (int i = 0; i < S.size(); ++i) {
8         for (int j = i + 1; j < S.size(); ++j) {
9           QSet<QString> disjunct1 = splitDisjunct(S[i]);//
10          QSet<QString> disjunct2 = splitDisjunct(S[j]);
11          bool foundContradiction = false;
12          QString literalToRemove;
13          QString negatedLiteralToRemove;
14          for (const QString& literal : disjunct1) {
15            QString negatedLiteral = (literal.startsWith("!") ? literal.
                at(1) : "!" + literal);
16            if (disjunct2.contains(negatedLiteral)) {
17              foundContradiction = true;
18              literalToRemove = literal;
19              negatedLiteralToRemove = negatedLiteral;
20              break;
21            }
22          }
23          if (foundContradiction) {//
24            QSet<QString> resolvent = disjunct1;
25            resolvent.unite(disjunct2);
26            resolvent.remove(literalToRemove);
27            resolvent.remove(negatedLiteralToRemove);
28            if (resolvent.isEmpty()) {
29              if (!S.contains("0")) {
30                newDisjuncts.append("0");
31                newDisjunctAdded = true;
32              }
33            } else {
34              QString resolventStr = QStringList(resolvent.toList()).
                  join("+");
35              removeDuplicateLiterals(resolventStr);
36              if (!S.contains(resolventStr) && hasNoContradiction(
                  resolventStr)) {
```

```
37                      newDisjuncts.append(resolventStr);
38                      newDisjunctAdded = true;
39                    }
40                  }
41                }
42              }
43            }
44            newDisjuncts.removeDuplicates();
45            S.append(newDisjuncts);
46          }
47        }
```

Листинг 2. Метод void applyResolution().

Вывод результата производится с помощью метода `void printResult(QString t)`. Если множество S после применения правила резолюций содержит пустой дизъюнкт «0», то пользователю сообщается, что исходная теорема является логическим следствием введенных теорем, в противном случае теорема не является логическим следствием.

ГЛАВА 3. ТЕСТИРОВАНИЕ

§ 3.1. Введение

Для проверки работоспособности программы были разработаны следующие сценарии тестирования:

1. Ввод некорректоной формулы.
2. Ввод корректных формул допущений, и попытка выполнения метода резолюций при отсутствии гипотезы теоремы.
3. Ввод гипотезы теоремы, и выполнение метода резолюций при пустом множестве допущений.
4. Попытка ввода нескольких теорем.
5. Проверка верной теоремы методом резолюций.
6. Проверка ложной теоремы методом резолюций.
7. Ввод нескольких одинаковых допущений.

§ 3.2. Сценарий тестирования 1

При попытке сохранить формулу на рисунке 5 пользователю выдается следующее сообщение (рисунок 6)

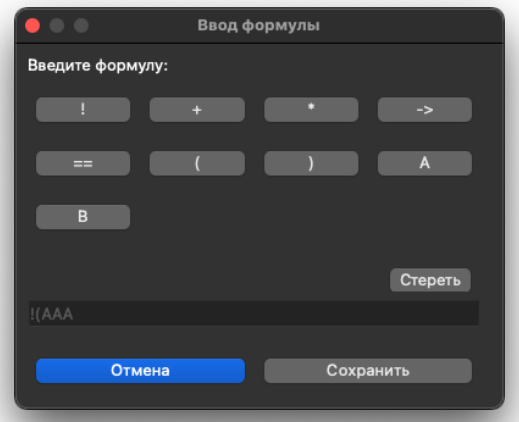

Рис. 5. Ввод некорректной формулы

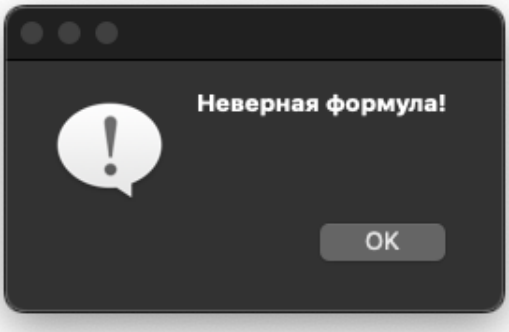

Рис. 6. Ошибка ввода неправильной формулы

Также отметим, что при попытке ввода двух операций подряд из множества `+`, `*`, `==`, `->`.

§ 3.3. Сценарий тестирования 2

Пусть введены верные допущения, но не введена гипотеза, как показано на рисунке 7.

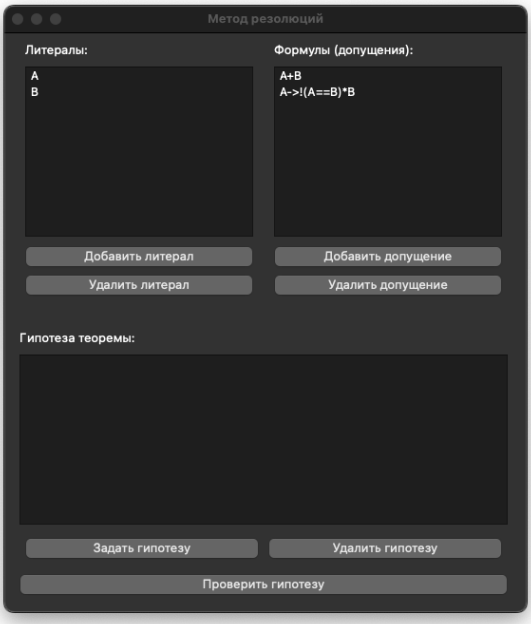

Рис. 7. Отсутствие гипотезы теоремы

В таком случае при попытке проверить гипотезу пользователь получит сообщение, показанное на рисунке 8.

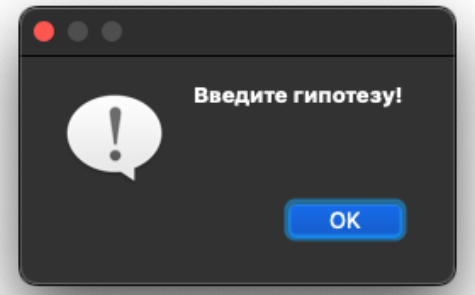

Рис. 8. Ошибка

§ 3.4. Сценарий тестирования 3

Пусть введена только гипотеза, а множество допущений не содержит формул, как показано на рисунке 9.

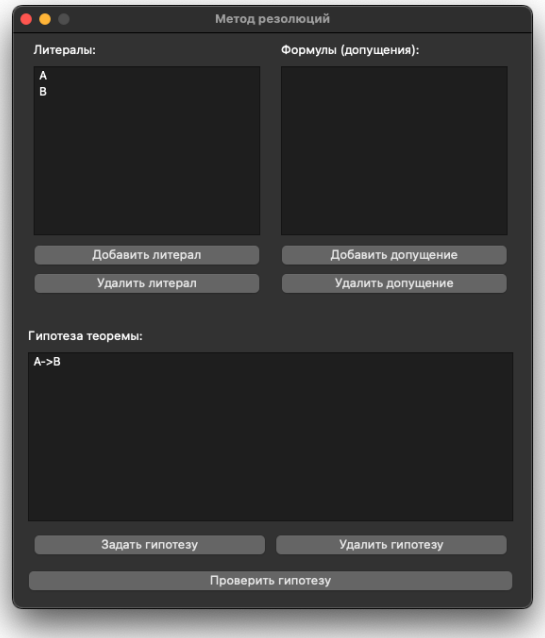

Рис. 9. Отсутствие допущений

В таком случае, так как из пустого множества формул ничего не следует, мы должны получить вывод о том, что введенная гипотеза не является логическим следствием пустого множества формул. Вывод приложения показан на рисунке 10.

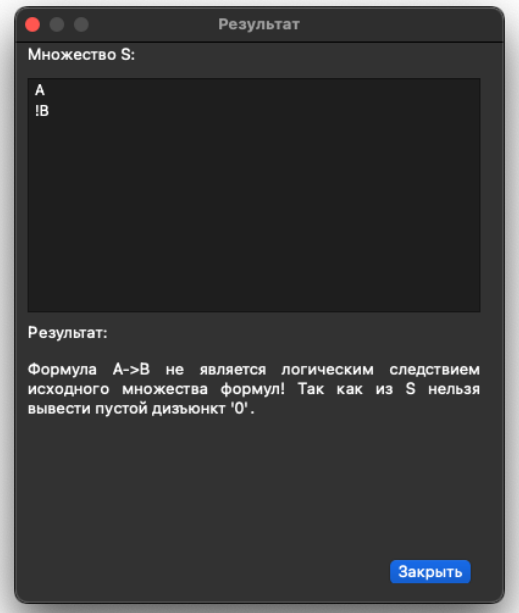

Рис. 10. Ошибочная гипотеза

§ 3.5. Сценарий тестирования 4

Пусть гипотеза теоремы уже введена (как показано на рисунке 9) и пользователь пробует ввести еще одну теорему.

В таком случае пользователь увидет следующую ошибку (рисунок 11).

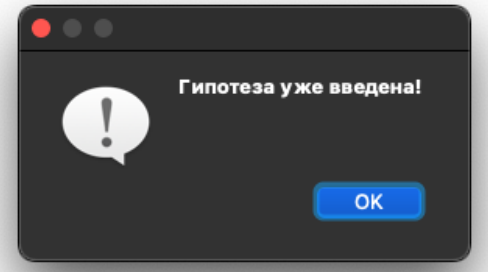

Рис. 11. Попытка ввести несколько теорем

§ 3.6. Сценарий тестирования 5

Пусть заданы формулы и теорема, такие как показано на рисунке 12.

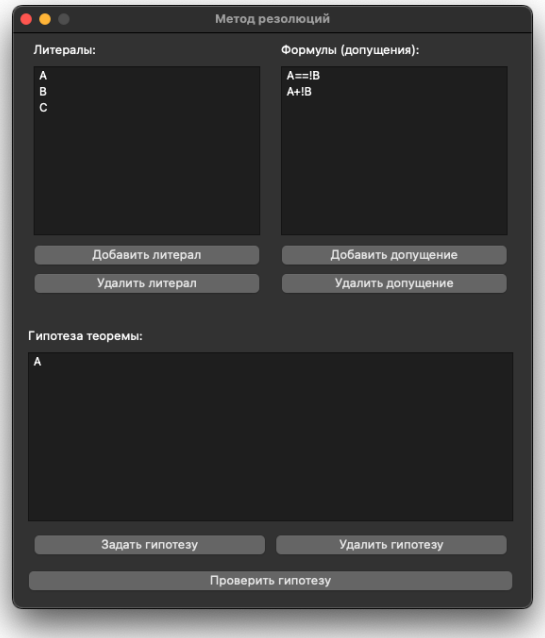

Рис. 12. Проверка верной теоремы

Рассмотрим результат метода резолюций (рисунок 13).

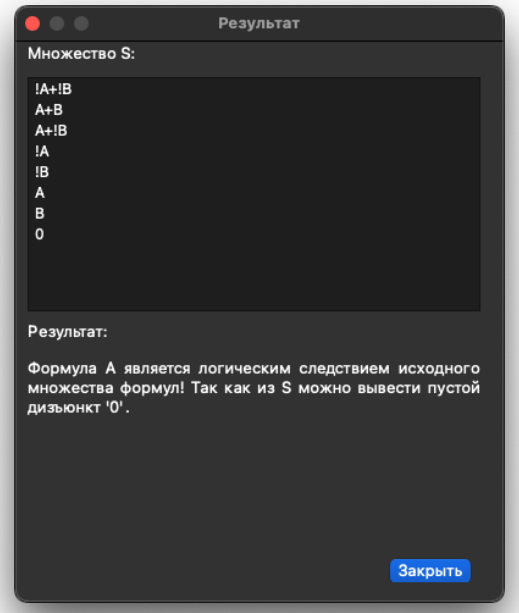

Рис. 13. Положительный результат

Рассмотрим получившееся множество S. Первые два дизъюнкта получились из КНФ первой формулы $A == !B = (!A + !B) * (A + !!B) = (!A + !B) * (A + B)$. Третий дизъюнкт получили из второй формулы (она уже находится в КНФ). Четверый дизъюнкт

- отрицание теоремы. Применив правило резолюций к третьему и четвертому дизъюнктам (A+!B и !A) получим пятый дизъюнкт (!B). Аналогично из пятого и второго дизъюнктов (!B и $A + B$) получим шестой A. Из второго и четвертого дизъюнктов ($A + B$ и !A) получим седьмой B. Пустой дизъюнкт можно получить из четвертого и шестого или из седьмого и пятого. Таким образом, формула A действительно является логическим следствием формул
A ==!B и A+!B.

§ 3.7. Сценарий тестирования 6

Пусть введены формулы, как показано на рисунке 14.

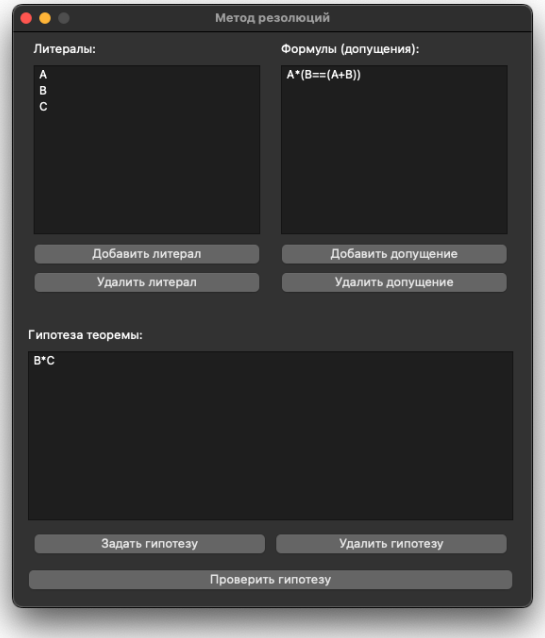

Рис. 14. Проверка ложной теоремы

Рассмотрим результат метода резолюций (рисунок 15).

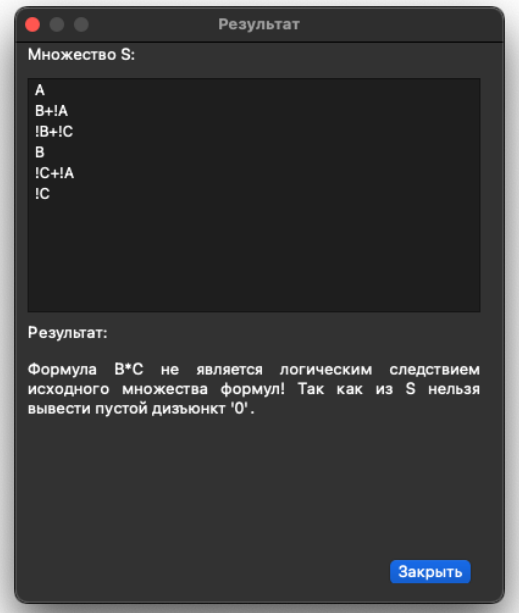

Рис. 15. Отрицательный результат

Рассмотрим множество дизъюнктов S. Первые два дизъюнкта получены из КНФ введенной формулы $A * (B == (A + B)) = A * (!B + A + B) * (B+!A) * (B+!B) = A*(B+!A)$. Третий дизъюнкт получается из КНФ отрицания теоремы $!(B*C) = !B+!C$. Далее из первого и второго дизъюнктов выводится четвертый B. Из второго $B+!A$ и третьего $!B+!C$ выводится пятый. Из первого и пятого выводится $!C$. Дальнейший вывод невозможен, а множество S не содержит пустого дизъюнкта, следовательно $B*C$ не следует из $A * (B == (A + B))$.

§ 3.8. Сценарий тестирования 7

Пусть пользователь уже ввел формулу $(B \rightarrow !A) == (C+(D \rightarrow !B))$, как показано на рисунке 16, и попробует ввести ее еще раз.

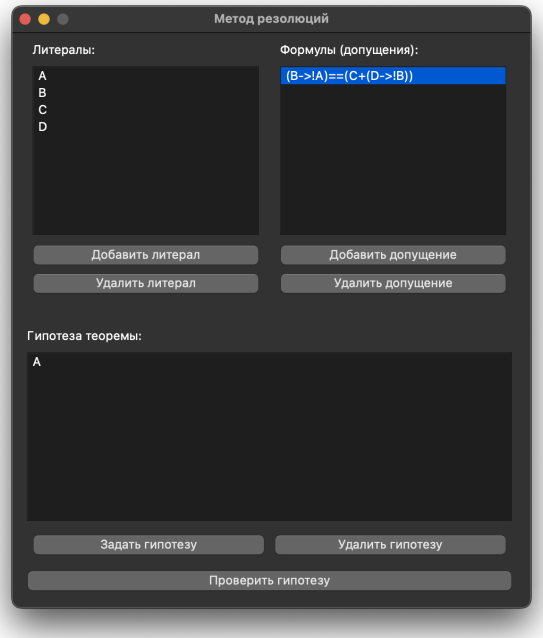

Рис. 16. Ввод формулы

В таком случае пользователь увидит следующие сообщение:

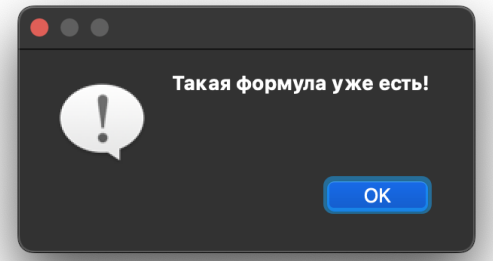

Рис. 17. Повторный ввод формулы

Проверим, какие дизъюнкты получились из введенной формулы (рисунок 18):

Приведем формулу $(B \to !A) == (C + (D \to !B))$ к КНФ: $(B \to !A) == (C + (D \to \to !B)) = (!B + !A) == (C + !D + !B) = (C + !D + !B + A) * (C + !D + !B + B) * (!B + !A + !C) * * (!B + !A + D + B) * (!B + !A + D) = (C + !D + !B + A) * (!B + !A + !C) * (!B + !A + D)$. Видим, что первые три дизъюнкта из множества S на рисунке 18 соответствуют КНФ данной формулы. Четвертый дизъюнкт - КНФ отрицания введенной теоремы, а дальше идет вывод из этих дизъюнктов.

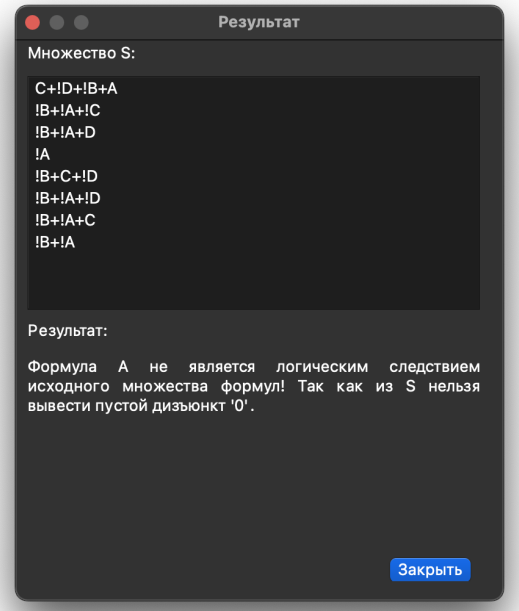

Рис. 18. Результат

§ 3.9. Итоги тестирования

Результаты тестирования показали, что программа успешно выполняет все заявленные функции. Все тесты были пройдены успешно, что подтверждает корректность работы алгоритмов.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы был изучен метод резолюций в логике высказываний, который является фундаментальным инструментом для автоматического доказательства теорем и логического вывода. Были исследованы теоретические основы метода резолюций, его алгоритмы и примеры применения.

В результате проведенного исследования была разработана программа, успешно реализующая метод резолюций. Программа состоит из трех основных классов: `MainWindow`, `Formula` и `Resolution`. Класс `MainWindow` обеспечивает пользовательский интерфейс, позволяющий пользователю вводить логические формулы и проверять теоремы методом резолюций. Классы `Formula` и `Resolution` реализуют основные алгоритмы работы с логическими формулами и выполнение резолюций соответственно.

Программа прошла тестирования различными сценариями. Результаты тестирования подтвердили корректность работы программы.

СПИСОК ИСТОЧНИКОВ

1. Замятин, А. П. Математическая логика и теория алгоритмов / А. П. Замятин - Екатеренбург, 2008. - 274 с.

2. Чень, Ч. Математическая логика и автоматическое доказательство теорем / Ч. Чень, Р. Ли; пер. с англ. Г. В. Давыдов, Г. Е. Минц, А. В. Сочилина - Москва, 1983. - 360 с.

3. Конъюнктивная нормальная форма - URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%8A%D1%8E%D0%BD%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0(Дата обращения: 06.07.2024)

4. Правило резолюций - URL: https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D0%BE_%D1%80%D0%B5%D0%B7%D0%BE%D0%BB%D1%8E%D1%86%D0%B8%D0%B9 (Дата обращения: 01.07.2024)

5. Разбор арифметического(и не только) выражения. Классические алгоритмы. - URL - https://algolist.ru/syntax/parsear.php (Дата обращения: 30.06.2024)

6. Создание UML-диаграмм - URL - https://app.diagrams.net/ (Дата обращения: 03.07.2024)

7. GitHub - URL - https://github.com/Vadikkkkk/A-resolution-technique/tree/release (Дата обращения: 08.07.2024)

Приложение

Файл MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSet>
#include <QMessageBox>

#include "formula.h"
#include "resolution.h"


QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
  Q_OBJECT

  public:
  MainWindow(QWidget *parent = nullptr);
  ~MainWindow();

  private:
  Ui::MainWindow* ui;

  int count;//
  QVector<QString> literals;//
  QVector<QString> formulas;//
  QString theorem;//

  QString lastButton;

  formula* addFormula;
  resolution* res;

  void updateList();//
  void updateFormulas();//
  void updateTheorem();//

  void connectAllButtons();


  public slots:
  void addLiteral();
  void deleteLiteral();

  void showAddFormula();
  void closeAddFormula();

  void saveFormula();
  void deleteFormula();

  void deleteTheorem();

```

```
56      void showResolution();
57      void closeResolution();
58
59  };
60  #endif // MAINWINDOW_H
```

Листинг 3. Файл MainWindow.h

Файл MainWindow.cpp

```
1       #include "mainwindow.h"
2       #include "ui_mainwindow.h"
3
4
5       MainWindow::MainWindow(QWidget *parent)
6       : QMainWindow(parent)
7       , ui(new Ui::MainWindow), count(0)
8       {
9         ui->setupUi(this);
10
11        addFormula = new formula;
12        res = new resolution;
13
14        connectAllButtons();
15      }
16
17      MainWindow::~MainWindow()
18      {
19        delete ui;
20        delete addFormula;
21        delete res;
22      }
23
24      void MainWindow::updateList()
25      {
26        ui->literals->clear();
27        for (const QString& str : literals) {
28          ui->literals->addItem(str);
29        }
30      }
31
32      void MainWindow::updateFormulas()
33      {
34        ui->formulas->clear();
35        for (const QString& str : formulas) {
36          ui->formulas->addItem(str);
37        }
38      }
39
40      void MainWindow::updateTheorem()
41      {
42        ui->theorem->clear();
43        ui->theorem->addItem(this->theorem);
44      }
45
46      void MainWindow::connectAllButtons()
47      {
48        connect(ui->add_literal, &QPushButton::clicked, this, &MainWindow
              ::addLiteral);
49        connect(ui->add_formula, &QPushButton::clicked, this, &MainWindow
              ::showAddFormula);
50        connect(ui->addTheorem,&QPushButton::clicked, this, &MainWindow::
```

```
            showAddFormula);
51
52          connect(ui->delete_literal, &QPushButton::clicked, this, &
                MainWindow::deleteLiteral);
53          connect(ui->delete_formula, &QPushButton::clicked, this, &
                MainWindow::deleteFormula);
54          connect(ui->deleteTheorem, &QPushButton::clicked, this, &
                MainWindow::deleteTheorem);
55          connect(ui->resolution, &QPushButton::clicked, this, &MainWindow::
                showResolution);
56
57          connect(addFormula, &formula::closeSignal, this, &MainWindow::
                closeAddFormula);
58          connect(addFormula, &formula::saveSignal, this, &MainWindow::
                saveFormula);
59
60          connect(res, &resolution::closeSignal, this, &MainWindow::
                closeResolution);
61        }
62
63        void MainWindow::addLiteral()
64        {
65          if(count < 26){
66            QChar letter = 'A';
67            literals.append(QString(letter.unicode() + count));
68            count++;
69            updateList();
70          }
71          else{
72            QMessageBox::information(this, "             !", "
                                          !");
73          }
74        }
75
76        void MainWindow::deleteLiteral()
77        {
78          if(count > 0){
79            literals.pop_back();
80            count--;
81            updateList();
82          }
83          else{
84            QMessageBox::information(this, "             !", "
                                        !");
85          }
86        }
87
88        void MainWindow::showAddFormula()
89        {
90          QPushButton *button = qobject_cast<QPushButton *>(sender());
91          if (button == ui->addTheorem && theorem.isEmpty()) {
92            lastButton = ui->addTheorem->text();
93            setEnabled(false);
94            addFormula->clearLine();
95            addFormula->braketsCount = 0;
96            addFormula->l = literals;
97            addFormula->updateButtons();
98            addFormula->show();
99          }
100         else if (button == ui->addTheorem && !theorem.isEmpty()){
```

```
101           QMessageBox::information(this, "              !", "
                                                  !");
102         }
103         else if (button == ui->add_formula) {
104           lastButton = ui->add_formula->text();
105           setEnabled(false);
106           addFormula->clearLine();
107           addFormula->braketsCount = 0;
108           addFormula->l = literals;
109           addFormula->updateButtons();
110           addFormula->show();
111         }
112
113       }
114
115       void MainWindow::closeAddFormula()
116       {
117         setEnabled(true);
118         addFormula->reject();
119       }
120
121       void MainWindow::saveFormula()
122       {
123         if (lastButton == ui->addTheorem->text()){
124           if (!addFormula->getFormula().isEmpty()){
125             theorem = addFormula->getFormula();
126             updateTheorem();
127             setEnabled(true);
128             addFormula->reject();
129           }
130           else{
131             QMessageBox::information(this, "              !", "
                                              !");
132           }
133         }
134         else if(lastButton == ui->add_formula->text()){
135           if (!addFormula->getFormula().isEmpty() && !formulas.contains(
                addFormula->getFormula())){
136             formulas.append(addFormula->getFormula());
137             updateFormulas();
138             setEnabled(true);
139             addFormula->reject();
140           }
141           else if(formulas.contains(addFormula->getFormula())){
142             QMessageBox::information(this, "              !", "
                                                !");
143             showAddFormula();
144           }
145           else{
146             QMessageBox::information(this, "              !", "
                                              !");
147             setEnabled(true);
148             addFormula->reject();
149           }
150         }
151       }
152
153       void MainWindow::deleteFormula()
154       {
155         if(formulas.size() > 0){
```

```
156           formulas.pop_back();
157           updateFormulas();
158         }
159         else{
160           QMessageBox::information(this, "            !", "
                    !");
161         }
162       }
163
164       void MainWindow::deleteTheorem()
165       {
166         theorem.clear();
167         ui->theorem->clear();
168       }
169
170       void MainWindow::showResolution()
171       {
172         if (ui->theorem->count()){
173           setEnabled(false);
174           res->literals = this->literals;
175           res->formulas = this->formulas;
176           res->theorem = this->theorem;
177           res->S.clear();
178           res->update();
179           res->show();
180         }
181         else{
182           QMessageBox::information(this, "            !", "
                          !");
183         }
184       }
185
186       void MainWindow::closeResolution()
187       {
188         setEnabled(true);
189         res->reject();
190       }
```

Листинг 4. Файл MainWindow.cpp

Файл Formula.h

```
1
2       #ifndef FORMULA_H
3       #define FORMULA_H
4
5       #include <QDialog>
6       #include <QSet>
7       #include <QString>
8       #include <QKeyEvent>
9
10      namespace Ui {
11        class formula;
12      }
13
14      class formula : public QDialog
15      {
16        Q_OBJECT
17
18        public:
```

```
19          explicit formula(QWidget *parent = nullptr);
20          ~formula();
21          QVector<QString> l;//
22          int braketsCount;
23
24          void updateButtons();
25          void addButtonsToLayout(const QVector<QString> &vector);
26          QString getFormula();
27          void clearLine();
28          bool check();
29
30          private:
31          Ui::formula *ui;
32          int row, column;//
33
34          QVector<QString> buttons;
35
36          void connectAllButtons();
37
38          public slots:
39          void close();
40          void onButtonClicked(const QString &text);
41          void backspace();
42
43          void save();
44
45          signals:
46          void closeSignal();
47          void saveSignal();
48          void backspacePressed();
49
50          protected:
51          void closeEvent(QCloseEvent* event) override;
52          void keyPressEvent(QKeyEvent *event) override;
53
54      };
55
56      #endif // FORMULA_H
```

Листинг 5. Файл Formula.h

Файл Formula.h

```
1       #include "formula.h"
2       #include "ui_formula.h"
3       #include <QMessageBox>
4
5       formula::formula(QWidget *parent) :
6       QDialog(parent), row(0), column(0), braketsCount(0),
7       ui(new Ui::formula)
8       {
9           ui->setupUi(this);
10
11          connectAllButtons();
12      }
13
14
15      void formula::addButtonsToLayout(const QVector<QString> &vector)
16      {
17          for (const QString &buttonText : vector) {
```

```
18          QPushButton *button = new QPushButton(buttonText, this);
19          connect(button, &QPushButton::clicked, this, [this, buttonText
               ]() {
20            onButtonClicked(buttonText);
21          });
22          ui->gridLayout->addWidget(button, row, column);
23          column++;
24          if (column >= 4) {
25            column = 0;
26            row++;
27          }
28        }
29      }
30
31      QString formula::getFormula()
32      {
33        return ui->lineEdit->text();
34      }
35
36      void formula::clearLine()
37      {
38        ui->lineEdit->clear();
39      }
40
41      bool formula::check()
42      {
43        auto isPartOfSet = [](const QString &ch, const QVector<QString> &
             set) -> bool {
44          return set.contains(ch);
45        };
46
47        QVector<QString> specialSet = {"*", "+", "->", "=="};
48        QString str = ui->lineEdit->text();
49        int eq = 0;
50
51
52        if(braketsCount || str[0] == "+" || str[0] == "*") return false;
53
54        if(str.contains("-") && !str.contains("->")) return false;
55
56        for (int i = 0; i < str.length(); ++i) {
57          // Check if the current character and the next character are
               both in l
58          if (i < str.length() - 1 && isPartOfSet(str.mid(i, 1), l) &&
               isPartOfSet(str.mid(i + 1, 1), l)) {
59            return false;
60          }
61          if (str[i] == "=") eq++;
62
63
64          // Check if the current character and the next character are
               both in specialSet
65          for (const QString &special : specialSet) {
66            if (i < str.length() - special.length() && str.mid(i, special.
                 length()) == special &&
67            str.mid(i + special.length(), special.length()) == special) {
68              return false;
69            }
70            if (str.endsWith(special) || str.endsWith("!")) return false;
71          }
```

```
72          }
73          if(eq % 2) return false;
74
75          return true;
76        }
77
78        void formula::connectAllButtons()
79        {
80          connect(ui->close_button, &QPushButton::clicked, this, &formula::
                close);
81          connect(ui->backspace, &QPushButton::clicked, this, &formula::
                backspace);
82          connect(ui->save_button, &QPushButton::clicked, this, &formula::
                save);
83          connect(this, &formula::backspacePressed, this, &formula::
                backspace);
84
85          buttons << "!" << "+" << "*" << "->" << "==" << "(" << ")";
86          addButtonsToLayout(buttons);
87        }
88
89
90        formula::~formula()
91        {
92          delete ui;
93        }
94
95        void formula::updateButtons()
96        {
97          QLayoutItem *item;
98          while ((item = ui->gridLayout->takeAt(0)) != nullptr) {
99            delete item->widget();
100           delete item;
101         }
102         row = 0;
103         column = 0;
104
105         addButtonsToLayout(buttons);
106         addButtonsToLayout(l);
107       }
108
109
110
111       void formula::close()
112       {
113         emit closeSignal();
114       }
115
116       void formula::onButtonClicked(const QString &text)
117       {
118         QString currentText = ui->lineEdit->text();
119         QString operations = "+*->==!(";
120
121         if(text == "(" && !currentText.isEmpty()){
122           if(!l.contains(currentText.at(currentText.size() - 1))){
123             braketsCount++;
124             ui->lineEdit->setText(currentText + text);
125           }
126           else{
127
```

```
128           }
129         }
130         else if (text == "(" && currentText.isEmpty()){
131           braketsCount++;
132           ui->lineEdit->setText(currentText + text);
133         }
134         else if (text == ")" && braketsCount == 0){
135           QMessageBox::information(this, "            !", "
                                          !");
136         }
137         else if (text == ")" && braketsCount > 0) {
138           if(braketsCount > 0 && (currentText.at(currentText.size() - 1)
                == ')' ||
139           l.contains(currentText.at(currentText.size() - 1)))){
140             braketsCount--;
141             ui->lineEdit->setText(currentText + text);
142           }
143           else{
144
145           }
146         }
147         else if (currentText.isEmpty() && !l.contains(text) && text != "!"
               ){
148
149         }
150         else if (text == "+" || text == "->" || text == "*" || text == "=="
               ") {
151           if(operations.contains(currentText.at(currentText.size() - 1))){
152
153           }
154           else{
155             ui->lineEdit->setText(currentText + text);
156           }
157         }
158         else {
159           ui->lineEdit->setText(currentText + text);
160         }
161       }
162
163       void formula::backspace()
164       {
165         QString currentText = ui->lineEdit->text();
166         if(!currentText.isEmpty()){
167           if(currentText.at(currentText.length() - 1) == ')') braketsCount
                ++;
168           if(currentText.at(currentText.length() - 1) == '(') braketsCount
                --;
169           currentText.chop(1);
170           ui->lineEdit->clear();
171           ui->lineEdit->setText(currentText);
172         }
173         else{
174           QMessageBox::information(this, "            !", "
                        !");
175         }
176       }
177
178       void formula::save()
179       {
180         if(check()){
```

```
181           emit saveSignal();
182         }
183         else{
184           QMessageBox::information(this, "             !", "
                                              !");
185         }
186
187       }
188
189       void formula::closeEvent(QCloseEvent *event)
190       {
191         emit closeSignal();
192       }
193
194       void formula::keyPressEvent(QKeyEvent *event)
195       {
196         if (event->key() == Qt::Key_Backspace) {
197           emit backspacePressed();
198         }
199         else if (event->key() == Qt::Key_Return){
200           emit saveSignal();
201         }
202         else if (event->key() == Qt::Key_Escape){
203           emit closeSignal();
204         }
205         else {
206           QWidget::keyPressEvent(event);
207         }
208       }
```

Листинг 6. Файл Formula.cpp

Файл Resolution.h

```
1       #ifndef RESOLUTION_H
2       #define RESOLUTION_H
3
4       #include <QDialog>
5
6
7       namespace Ui {
8         class resolution;
9       }
10
11      class resolution : public QDialog
12      {
13        Q_OBJECT
14
15        public:
16        explicit resolution(QWidget *parent = nullptr);
17        ~resolution();
18        friend class MainWindow;
19
20        void update();
21
22        private:
23        Ui::resolution *ui;
24        QVector<QString> literals;//
25        QVector<QString> formulas;//
26        QStringList S;//
```

```
27          QString theorem;
28
29
30          QSet<QString> splitDisjunct(const QString& str);
31
32
33
34          void formulaToS(const QString& str);
35          void applyResolution();
36
37          void removeBrackets(QString& str);
38
39          void printResult(QString t);
40
41          //
42          void removeImplication(QString& str);//
                ->
43          void removeEq(QString& str);//                                  ==
44
45          void removeNotBrackets(QString& str);//
                !(
46
47          void removeDoubleNot(QString& str);//
48          void replaceParentheses(QString& str);//
49
50          void ToCNF(QString& str);
51
52          bool hasNoContradiction(const QString &formula);
53          void removeDuplicateLiterals(QString &formula);
54
55
56          public slots:
57          void close();
58
59          signals:
60          void closeSignal();
61
62          protected:
63          void closeEvent(QCloseEvent* event) override;
64      };
65
66      #endif // RESOLUTION_H
```

Листинг 7. Файл Resolution.h

Файл Resolution.cpp

```
1       #include "resolution.h"
2       #include "ui_resolution.h"
3
4       resolution::resolution(QWidget *parent) :
5       QDialog(parent),
6       ui(new Ui::resolution)
7       {
8           ui->setupUi(this);
9
10          connect(ui->close, &QPushButton::clicked, this, &resolution::
                closeSignal);
```

```
11
12        ui->result->setAlignment(Qt::AlignTop | Qt::AlignJustify);
13        ui->result->setWordWrap(true);
14      }
15
16      resolution::~resolution()
17      {
18        delete ui;
19      }
20
21      void resolution::update()
22      {
23        ui->formulas->clear();//
                                                          S
24        for (QString& str : formulas) {//

25          str.insert(0,'(');//
26          str.append(')');
27          ToCNF(str);
28          formulaToS(str);
29        }
30        QString tmp = theorem;
31        theorem.insert(0,"!(");
32        theorem.append(')');
33        ToCNF(theorem);
34        formulaToS(theorem);
35        applyResolution();
36        ui->formulas->addItems(S);
37        printResult(tmp);
38      }
39
40      QSet<QString> resolution::splitDisjunct(const QString &str)
41      {
42        return QSet<QString>::fromList(str.split("+"));
43      }
44
45      void resolution::applyResolution()
46      {
47        bool newDisjunctAdded = true;
48        while (newDisjunctAdded) {
49          newDisjunctAdded = false;
50          QStringList newDisjuncts;
51          for (int i = 0; i < S.size(); ++i) {
52            for (int j = i + 1; j < S.size(); ++j) {
53              QSet<QString> disjunct1 = splitDisjunct(S[i]);//

54              QSet<QString> disjunct2 = splitDisjunct(S[j]);
55              bool foundContradiction = false;
56              QString literalToRemove;
57              QString negatedLiteralToRemove;
58              for (const QString& literal : disjunct1) {
59                QString negatedLiteral = (literal.startsWith("!") ?
                      literal.at(1) : "!" + literal);
60                if (disjunct2.contains(negatedLiteral)) {
61                  foundContradiction = true;
62                  literalToRemove = literal;
63                  negatedLiteralToRemove = negatedLiteral;
64                  break;
65                }
66              }
```

```
67              if (foundContradiction) {//
68                QSet<QString> resolvent = disjunct1;
69                resolvent.unite(disjunct2);
70                resolvent.remove(literalToRemove);
71                resolvent.remove(negatedLiteralToRemove);
72                if (resolvent.isEmpty()) {
73                  //                ,          resolvent             ,
                                                        "0"
74                  if (!S.contains("0")) {
75                    newDisjuncts.append("0");
76                    newDisjunctAdded = true;
77                  }
78                } else {
79                  QString resolventStr = QStringList(resolvent.toList()).
                      join("+");
80                  removeDuplicateLiterals(resolventStr);
81                  if (!S.contains(resolventStr) && hasNoContradiction(
                      resolventStr)) {
82                    newDisjuncts.append(resolventStr);
83                    newDisjunctAdded = true;
84                  }
85                }
86              }
87            }
88          }
89          newDisjuncts.removeDuplicates();
90          S.append(newDisjuncts);
91        }
92      }
93
94      void resolution::formulaToS(const QString &str)
95      {
96        if(!str.isEmpty()){
97          QStringList substrings = str.split('*');
98          for (QString& s : substrings) {
99            removeBrackets(s);
100           S.append(s);
101         }
102         S.removeDuplicates();
103       }
104
105     }
106
107     void resolution::removeBrackets(QString &str)
108     {
109       str.remove('(');
110       str.remove(')');
111     }
112
113
114     void resolution::printResult(QString t)
115     {
116
117       if (S.contains("0")){
118         ui->result->setText("               " + t + "
                                                        "
119         "                                                  !
                        S"
120         "                                                          '0'.")
              ;
```

```
    }
    else{
      ui->result->setText("                " + t + "                         "
      "                                         !
                  S "
      "                                                          '0'."
        );
    }
  }

  void resolution::removeImplication(QString &str)
  {
    while(str.contains("->")){
      for (int i = 0; i < str.size(); i++ ) {
        if(str.at(i) == '-'){//
          QString left;
          QString right;
          int indexToReplace;

          str.replace(i, 2, '+');//                    +
          if(str.at(i-1) == ')'){//
                                         ,
            //
            int closedBrackets = 1;
            int openBrackets = 0;
            left += ")";
            for (int j = i - 2;j >= 0 ; j--) {
              if (str.at(j) == ')') closedBrackets++;
              if (str.at(j) == '(') openBrackets++;
              left.insert(0, str.at(j));
              if (closedBrackets == openBrackets){//

                indexToReplace = j;
                //str.insert(j, "(!");
                break;
              }
            }
          }
          if(left.isEmpty()){//

            left += str.at(i-1);
            if(i-2 >= 0){
              if (str.at(i-2) == '!'){
                left.insert(0, '!');
                indexToReplace = i-2;
              }
              else{
                indexToReplace = i-1;
              }
            }
            else{
              indexToReplace = i-1;
            }
          }
          if(str.at(i+1) == '('){//
            int closedBrackets = 0;
            int openBrackets = 1;
            right += "(";
```

```
174                 for (int j = i + 2; j < str.size(); j++ ) {//
175                   right.append(str.at(j));
176                   if (str.at(j) == ')') closedBrackets++;
177                   if (str.at(j) == '(') openBrackets++;
178                   if (openBrackets == closedBrackets){
179                     break;
180                   }
181                 }
182               }
183               if(str.at(i+1) == '!' && str.at(i+2) != '('){//
184                 right = '!' + str.at(i+2);
185               }
186               if(str.at(i+1) != '!' && str.at(i+1) != '('){//
187                 right += str.at(i+1);
188               }
189               if(str.at(i+1) == '!' && str.at(i+2) == '('){//
190                 int openBrackets = 1;
191                 int closedBrackets = 0;
192                 right += "!(";
193                 for (int j = i + 3; j < str.size(); j++ ) {//
194                   right.append(str.at(j));
195                   if (str.at(j) == ')') closedBrackets++;
196                   if (str.at(j) == '(') openBrackets++;
197                   if (openBrackets == closedBrackets){
198                     break;
199                   }
200                 }
201               }
202               QString newStr = "(!" + left + "+" + right + ")";
203               str.replace(indexToReplace, left.size() + 1 + right.size(),
                     newStr);
204               QRegularExpression re("!!");
205               str.replace(re, "");
206             }
207           }
208         }
209       }
210
211       void resolution::removeEq(QString &str)
212       {
213         while(str.contains("==")){
214           for (int i = 0; i < str.size(); i++){
215             if(str.at(i) == '='){//
216               QString left;
217               QString right;
218               int indexToReplace;
219               str.replace(i, 2, '*');
220               if(str.at(i-1) == ')'){//
221                 int closedBrackets = 1;
222                 int openBrackets = 0;
223
224                 left += ")";
225                 for (int j = i - 2;j >= 0 ; j--){//
```

```
226                     left.insert(0,str.at(j));//
227                     if (str.at(j) == ')') closedBrackets++;
228                     if (str.at(j) == '(') openBrackets++;
229
230                     if (closedBrackets == openBrackets){//
231                       indexToReplace = j;
232                       break;
233                     }
234                   }
235                 }
236                 if(str.at(i+1) == '('){//
237                   int openBrackets = 1;
238                   int closedBrackets = 0;
239                   right += "(";
240                   for (int j = i + 2; j < str.size(); j++ ) {//
241                     right.append(str.at(j));
242                     if (str.at(j) == ')') closedBrackets++;
243                     if (str.at(j) == '(') openBrackets++;
244                     if (openBrackets == closedBrackets){
245                       break;
246                     }
247                   }
248                 }
249                 if(str.at(i+1) == '!' && str.at(i+2) != '('){//
250                   right = '!' + str.at(i+2);
251                 }
252                 if(str.at(i+1) == '!' && str.at(i+2) == '('){//
253                   int openBrackets = 1;
254                   int closedBrackets = 0;
255                   right += "!(";
256                   for (int j = i + 3; j < str.size(); j++ ) {//
257                     right.append(str.at(j));
258                     if (str.at(j) == ')') closedBrackets++;
259                     if (str.at(j) == '(') openBrackets++;
260                     if (openBrackets == closedBrackets){
261                       break;
262                     }
263                   }
264                 }
265                 if (str.at(i+1) != '(' && str.at(i+1) != '!'){//
266                   right += str.at(i+1);
267                 }
268                 if(left.isEmpty()){//
269                   left += str.at(i-1);
270                   if(i-2 >= 0){
271                     if (str.at(i-2) == '!'){
272                       left.insert(0, '!');
273                       indexToReplace = i-2;
274                     }
275                     else{
276                       indexToReplace = i-1;
277                     }
```

```
278                 }
279                 else{
280                   indexToReplace = i-1;
281                 }
282               }
283               QString newStr = "((!" + left + "+" + right + ")*(" + left +
                      "+!" + right + "))";
284               str.replace(indexToReplace, left.size() + 1 + right.size(),
                      newStr);
285               QRegularExpression re("!!");
286               str.replace(re, "");
287             }
288           }
289         }
290       }
291
292       void resolution::removeNotBrackets(QString &str)
293       {
294         while(str.contains("!(")){
295           for (int i = 0; i < str.size(); i++) {
296             if(str.at(i) == '!'){//
297               QString subString;
298               QString left;
299               QString right;
300               QString operation;
301               int operationIndex = 0;
302               int endIndex;
303               if(str.at(i+1) != '('){//
                                        ,
304
305               }
306               else{//
307                 int openBrackets = 1;
308                 int closedBrackets = 0;
309                 int level = 0;
310                 for (int j = i + 2; i < str.size(); j++) {
311                   if (str.at(j) == '('){
312                     level++;
313                     openBrackets++;
314                   }
315                   if (str.at(j) == ')'){
316                     level--;
317                     closedBrackets++;
318                   }
319                   if(level == 0 && (str.at(j) == '+' || str.at(j) == '*'))
                          {
320                     if(str.at(j) == '+') operation = "*";
321                     else operation = "+";
322                     operationIndex = j;
323                   }
324                   subString.append(str.at(j));
325                   if (openBrackets == closedBrackets){
326                     subString.chop(1);
327                     endIndex = j;
328                     break;
329                   }
330                 }
331                 if(operationIndex != 0){//

332                   left = str.mid(i+2, operationIndex - i - 2);//
```

```
333                 right = str.mid(operationIndex + 1, endIndex -
                        operationIndex - 1);
334                 str.replace(i, endIndex - i + 1, "(!(" + left + ")" +
                        operation + "!(" + right + "))");
335               }
336               else{//
337                 str.remove(i+1, 1);
338                 str.remove(endIndex-1, 1);
339               }
340             }
341           }
342         }
343       }
344     }
345
346     void resolution::removeDoubleNot(QString &str)
347     {
348       QRegularExpression re("!!");
349       str.replace(re,"");
350     }
351
352     void resolution::replaceParentheses(QString &str)
353     {
354       QString previous = str;
355       do{
356         previous = str;
357
358         for(int i = 0; i < str.size(); i++){
359           int endIndex;
360           QString subString;
361           QString oprationsOnLevelZero;
362           int openBrackets = 1;
363           int closedBrackets = 0;
364           int level = 0;
365           int lastOpIndex = i + 1;
366           QStringList slag;
367           QStringList mul;
368           QString res;
369           QString lastOp;
370
371           if(str.at(i) == '('){
372             for(int j = i + 1; j < str.size(); j++){
373               subString.append(str.at(j));
374               if (str.at(j) == '('){
375                 level++;
376                 openBrackets++;
377               }
378               if (str.at(j) == ')'){
379                 level--;
380                 closedBrackets++;
381               }
382               if(level == 0 && str.at(j) == '+'){
383                 oprationsOnLevelZero.append('+');
384               }
385               if(level == 0 && str.at(j) == '*'){
386                 oprationsOnLevelZero.append('*');
387               }
388               if (openBrackets == closedBrackets){
389                 subString.chop(1);
```

```
390                     endIndex = j;
391                     break;
392                   }
393                 }
394                 if(oprationsOnLevelZero.contains('*') && !
                      oprationsOnLevelZero.contains('+')){//CNF
395                   str.replace(i, endIndex - i + 1, subString);
396                   continue;
397                 }
398                 if(oprationsOnLevelZero.isEmpty()){//

399                   str.replace(i, endIndex - i + 1, subString);
400                   continue;
401                 }
402                 if(oprationsOnLevelZero.contains('+') && !
                      oprationsOnLevelZero.contains('*')){//

403                   continue;
404                 }
405                 if(oprationsOnLevelZero.contains('*') &&
                      oprationsOnLevelZero.contains('+')){
406                   QString tmp;
407                   int level = 0;
408                   bool lastOpIsMul = false;
409                   for(int j = 0; j < subString.size(); j++){
410                     if(subString.at(j) == '(') level++;
411                     if(subString.at(j) == ')') level--;
412                     tmp.append(subString.at(j));
413                     if((subString.at(j) == '*' || lastOpIsMul) && level == 0
                          && (subString.at(j) == '*' || subString.at(j) == '+'
                         )){
414                       tmp.chop(1);
415                       mul.append(tmp);
416                       tmp.clear();
417                       lastOpIndex = j;
418                       if(subString.at(j) == '+'){
419                         lastOpIsMul = false;
420                       }
421                       if(subString.at(j) == '*'){
422                         lastOpIsMul = true;
423                       }
424                       continue;
425                     }
426                     if(subString.at(j) == '+' && level == 0){
427                       tmp.chop(1);
428                       slag.append(tmp);
429                       tmp.clear();
430                       lastOpIndex = j;
431                       lastOpIsMul = false;
432                     }
433                   }
434                   if(subString.at(lastOpIndex) == '+'){
435                     slag.append(subString.mid(lastOpIndex+1));
436                   }
437                   if(subString.at(lastOpIndex) == '*'){
438                     mul.append(subString.mid(lastOpIndex+1));
439                   }
440                   res = "(" + slag.join("+") + "+" + mul.at(0) + ")*(";
441                   mul.removeFirst();
442                   res += slag.join("+") + "+" + mul.join('*') + ")";
```

```
443               }
444               str.replace(i, subString.size()+2, res);
445               res.clear();
446               slag.clear();
447               mul.clear();
448             }
449           }
450         }while(str != previous);
451       }
452
453       void resolution::ToCNF(QString &str)
454       {
455         removeEq(str);
456         removeImplication(str);
457         removeNotBrackets(str);
458         removeDoubleNot(str);
459         replaceParentheses(str);
460
461         QStringList res;
462         QStringList tmp = str.split('*');
463         for (QString& s : tmp) {
464           removeBrackets(s);
465           if(hasNoContradiction(s)){
466             removeDuplicateLiterals(s);
467             res.append(s);
468           }
469         }
470
471         str = res.join('*');
472
473       }
474
475       bool resolution::hasNoContradiction(const QString &formula)
476       {
477         QSet<QString> literals;
478         QStringList parts = formula.split('+');
479         for (const QString &part : parts) {
480           QString literal = part.trimmed();
481           if (literal.startsWith('!')) {
482             //

483             QString positiveLiteral = literal.mid(1);
484             if (literals.contains(positiveLiteral)) {
485               return false;
486             }
487             literals.insert(literal);
488           } else {
489             //

490             QString negativeLiteral = '!' + literal;
491             if (literals.contains(negativeLiteral)) {
492               return false;
493             }
494             literals.insert(literal);
495           }
496         }
497         return true;
498       }
499
500       void resolution::removeDuplicateLiterals(QString &formula)
```

```
501       {
502         QSet<QString> uniqueLiterals;
503         QStringList parts = formula.split('+');
504         QStringList result;
505
506         for (const QString &part : parts) {
507           QString literal = part.trimmed();
508           if (!uniqueLiterals.contains(literal)) {
509             uniqueLiterals.insert(literal);
510             result.append(literal);
511           }
512         }
513
514         formula = result.join('+');
515       }
516
517
518
519       void resolution::close()
520       {
521         emit closeSignal();
522       }
523
524       void resolution::closeEvent(QCloseEvent *event)
525       {
526         emit closeSignal();
527       }
```

Листинг 8. Файл Resolution.cpp

Файл main.cpp

```
1       #include "mainwindow.h"
2
3
4       #include <QApplication>
5
6       int main(int argc, char *argv[])
7       {
8         QApplication a(argc, argv);
9         MainWindow w;
10        w.show();
11        return a.exec();
12      }
```

Листинг 9. Файл main.cpp