МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Вятский государственный университет»
(«ВятГУ»)
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Допущено к защите
Руководитель проекта
_____/Мельцов В.Ю./
(подпись) (Ф.И.О)
«__»_____2018г.

РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ ЛОГИЧЕСКОГО ВЫВОДА

Пояснительная записка
Курсовой проект по дисциплине
«Комплекс знаний бакалавра»
ТПЖА 09.03.01.024 ПЗ

Разработал студент группы ИВТ-42 _____/Щесняк Д.С./
Руководитель доцент кафедры ЭВМ_____/Мельцов В.Ю./

Проект защищен с оценкой «_____» _____
(оценка) (дата)

Члены комиссии _____/ _____/
(подпись) (Ф.И.О)

_____ /_____/

Киров 2018

Содержание

Введение

Обработка знаний – одна из сфер применения искусственного интеллекта, которая предполагает использование компьютером знаний для решения проблем и принятия решений. Переход от архитектуры, которая ориентирована на обработку данных, к архитектуре обработки знаний, является одной из ключевых проблем развития ЭВМ. Важную роль в обработке знаний играет логический вывод. С реализацией логического вывода связано три проблемы: разработка методов логического вывода, создание ПО для описания и решения поставленной задачи логическим выводом, аппаратная поддержка логического вывода.

В настоящее время реализовано множество программ логического вывода. В своей основе данные программы для решения задач используют метод резолюции. Существует другой метод, который за меньшее количество шагов позволяет выполнить логический вывод. Данный метод использует операцию деления дизъюнктов.

Наибольший интерес представляет логический вывод в логике предикатов первого порядка. Данный аппарат математической логики имеет хорошо изученную теоретическую базу и представляет большие возможности по сравнению с логикой высказываний.

Таким образом, разработка программного обеспечения логического вывода путем применения метода обобщенного деления дизъюнктов является перспективный направлением для развития данной области. Программная реализация позволит более детально изучить данный метод и рассмотреть возможные способы его развития.

1 Анализ предметной области

В данной главе производится анализ предметной области, обзор существующих программных решений поставленной задачи.

1.1 Анализ состояний проблемы

Исчисление высказываний ограничено структурой предложений в терминах простых высказываний, которых покрывают лишь малую часть высказываний. Исчисление предикатов вводит предикаты, которые позволяют строить высказывания вида: Джон любит вино.

Из высказываний составляется база знаний, которая используется в логическом выводе. Всего существует 3 основных вида логического вывода:

1. Дедукция. Дедукция предполагает выведение результата из общих правил и посылок.
2. Индукция. Предполагает выведение общего правила из заданных предпосылок и результата.
3. Абдукция – процесс выведения предпосылки из общего правила и результата.

Данные логические выводы можно наглядно проиллюстрировать на известном силлогизме:

1. Все люди смертны. ∀ x Человек(x) => Смертен(x)
2. Сократ является человеком. Человек(Сократ)
3. Сократ смертен. Смертен(Сократ)

В данном случае первое высказывание является правилом, второе – посылкой, а третье – выводом. Дедукция позволяет на основе высказываний, что все люди смертны и Сократ – человек, сделать вывод, что Сократ тоже смертен. Индукция же на основе того, что Сократ смертен и является человеком, позволяет сделать вывод, что все люди смертны. Абдукция выводит посылку о том, что Сократ человек на основе высказываний о смертности людей и Сократа.

Все методы логического вывода могут быть классифицированы по следующим критериям:

- По форме представления данных.
- По законам логики.
- По принципу логического вывода.
- По направленности. Методы характеризуются направлением вывода: от заключения к исходным посылкам, от исходных посылок к заключению, либо двунаправленный.
- По тактике обхода. Определяет по какому правилу будет обходиться дерево логического вывода.
- По стратегии. Выбор правильной стратегии позволяет повысить эффективность поиска необходимых дизъюнктов.

Классификация методов логического вывода представлена на рисунке 1.1.

В настоящее время существует множество методов логического вывода в логике предикатов. Большинство из них используют метод резолюций, который последовательно производит логический вывод, из-за чего процесс логического вывода может занять время.

Одним из наиболее перспективных методов, которые могут ускорить логический вывод является метод обобщенного деления дизъюнктов, в основе которого лежит стратегия вывода вширь, благодаря которой возможно делать параллельный вывод.

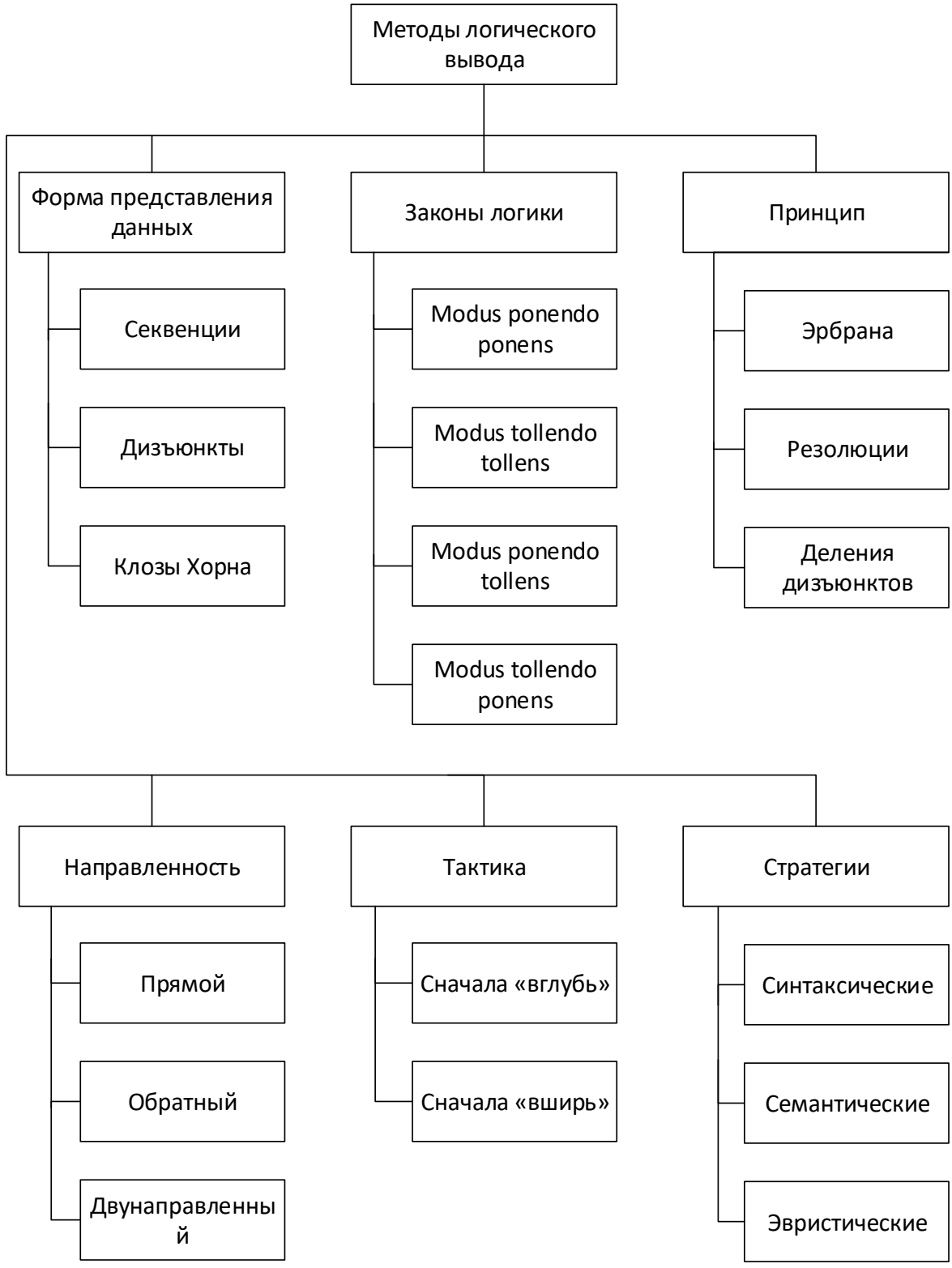

Рисунок 1.1 – Классификация методов логического вывода

1.2 Обзор аналогов

На данный момент имеется очень малое количество информации о программах логического вывода, использующих в своей основе метод деления дизъюнктов. Найденные программы были разработаны на кафедре ЭВМ ВятГУ и осуществляют логический вывод только в логике высказываний.

Логический вывод в исчислений высказываний

Данная программа была разработана на кафедре ЭВМ и представляет собой инструмент для логического вывода в исчислений высказываний.

Программа представляет собой приложение разработанное для операционной системы Windows, которая имеет следующие функциональные возможности:

- Дедуктивный и абдуктивный логический вывод методом деления дизъюнктов, а так же их модификации.
- Детальный отчет о логическом выводе с разбитием по шагам
- Преобразование исходных высказываний в вид дизъюнктов с последующим логическим выводом

Большой выбор методов логического вывода позволяет решать широкий круг задач. Однако функционал программы позволяет делать логический вывод только в логике предикатов, что существенно ограничивает множество доступных высказываний. Также существует ряд следующих недостатков:

- Отсутствие возможности ввода нескольких выводимых высказываний
- Отсутствие интерфейса, позволяющего редактировать набор высказываний загруженных из файла
- Игнорирование ошибок ввода исходных высказываний, что влечет за собой некорректную работу программы.

Пример графического интерфейса приложения представлен на рисунке 1.1. Пример выполнения абдуктивного логического вывода представлен на рисунке 1.2.

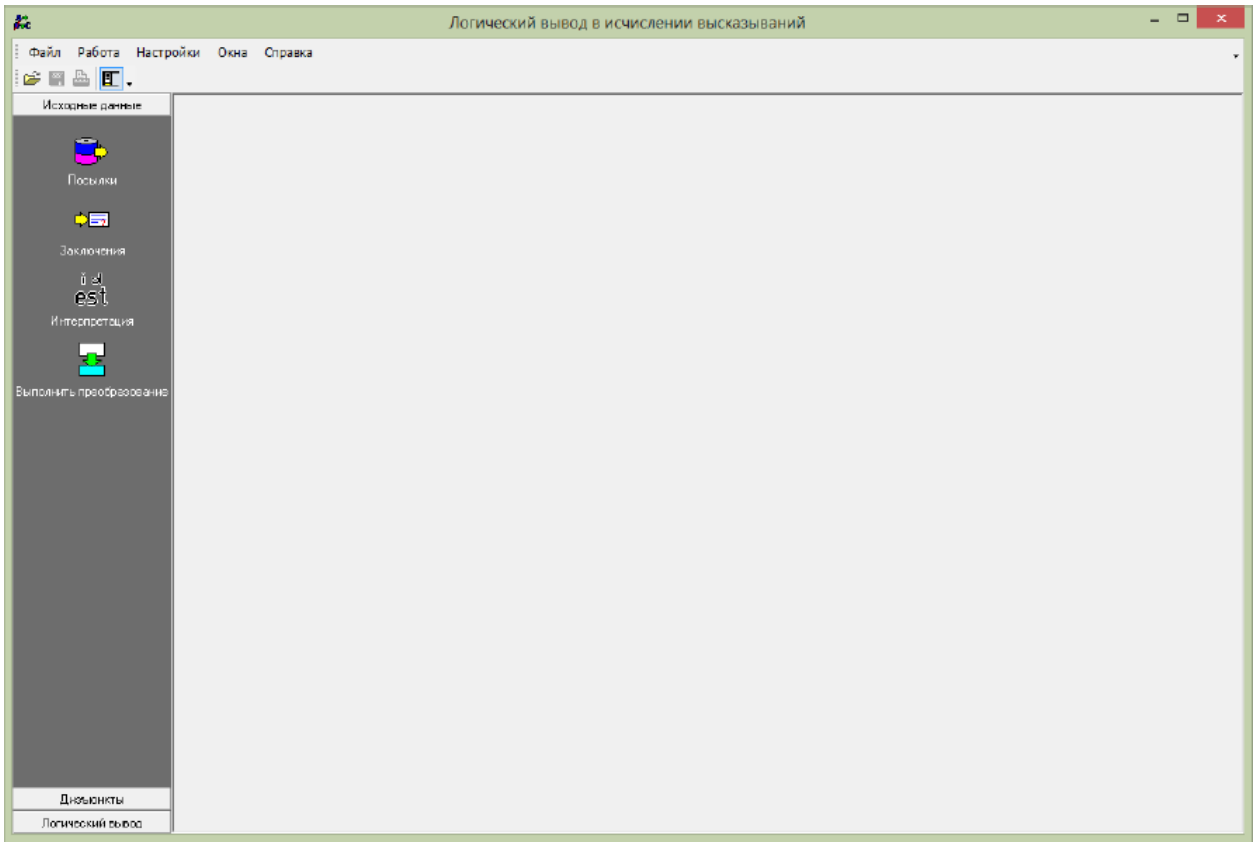

Рисунок 1.1 – Графический интерфейс программы логического вывода

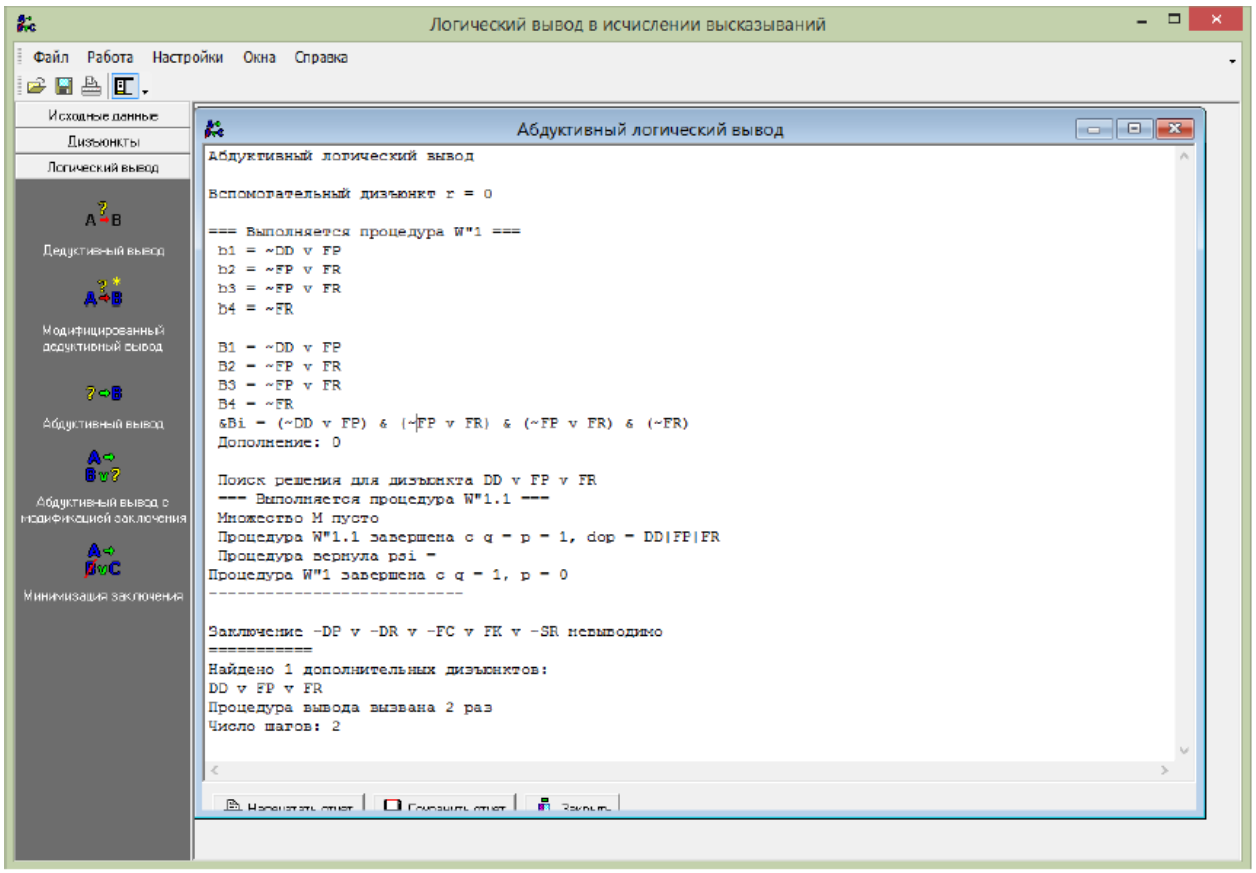

Рисунок 1.2 – Пример выполнения абдуктивного логического вывода

Lince

Данная программа была разработана в ходе дипломного проектирования студентами кафедры ЭВМ.

Программа представляет собой приложение разработанное на языке программирование Java для операционных систем семейства Windows, которая имеет следующие возможности:

- Логический вывод в логике высказываний методом деления дизъюнктов.
- Построение графической схемы вывода высказываний.
- Поддержка вывода нескольких высказываний.
- Дедуктивный и абдуктивный логический вывод.
- Вывод информации о выполняемых процедурах в журнал выполнения

Основным недостатком данной программы является логический вывод только в логике высказываний, что существенно ограничивает множество возможных высказываний.

Пример графического интерфейса представлен на рисунке 1.3.

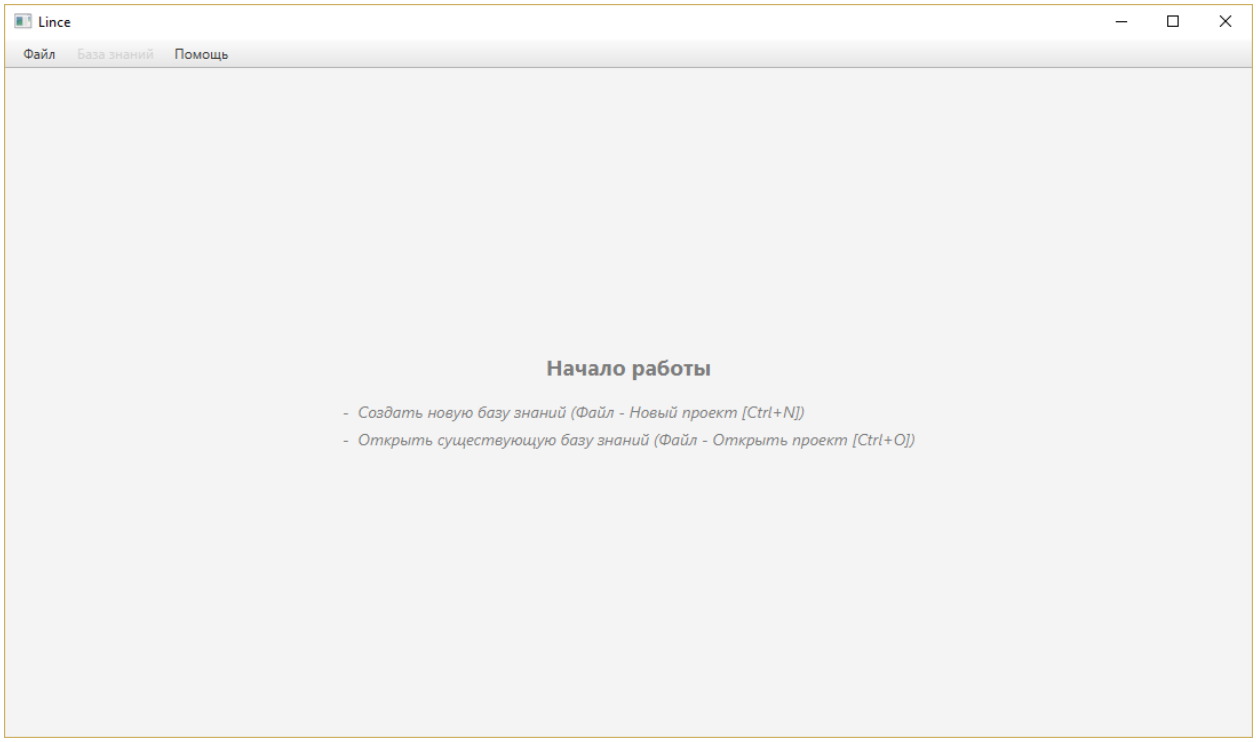

Рисунок 1.3 – Графический интерфейс приложения Lince

1.3 Анализ систем программирования

Visual prolog

Является объектно-ориентированной модификацией пролога со строгой типизацией. Логические языки программирования традиционно являются интерпретируемыми, но visual prolog является компилируемым, благодаря чему появляется возможность отлавливания синтаксических ошибок на стадии компиляции. На фоне остальных реализация выделяется возможностью удобного проектирования пользовательского графического интерфейса. Основным недостатком данной реализации является доступность только под операционной системой windows. Пример графического интерфейса представлен на рисунке 1.4.

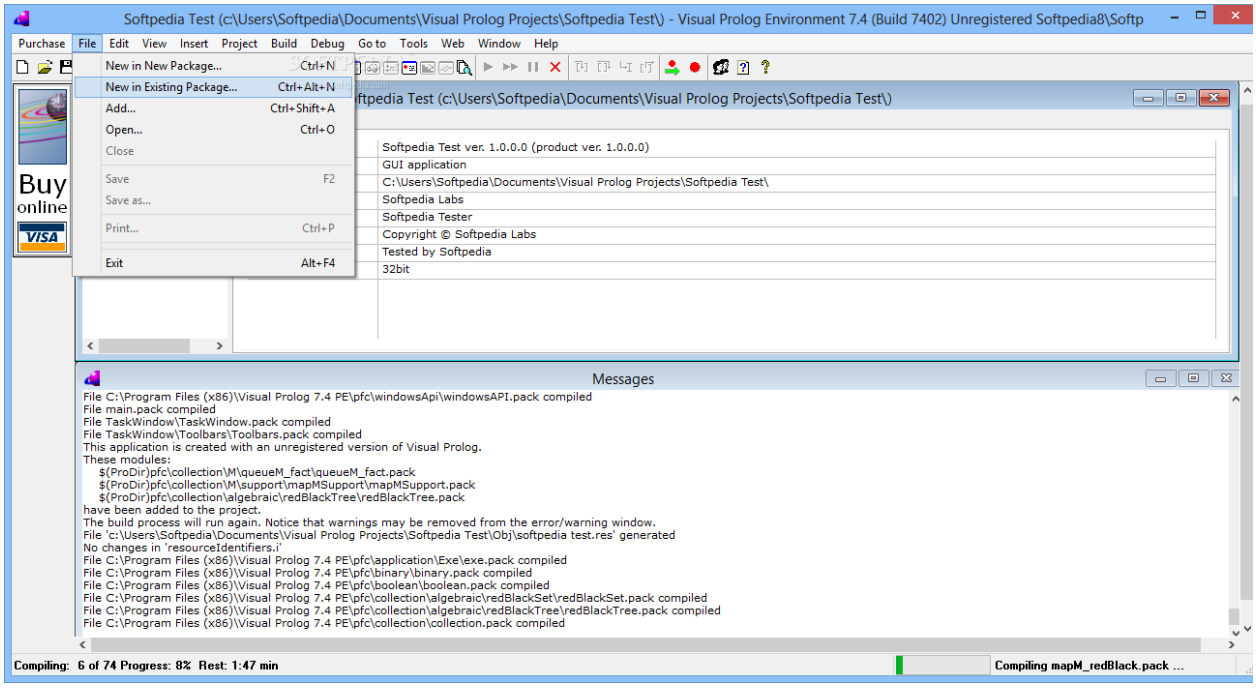

Рисунок 1.4 – Графический интерфейс Visual Prolog

Swi-prolog

Swi-prolog является свободной реализацией языка программирования пролог. Имеет богатый набор функций, библиотек для реализации параллельных вычислений, разработки графического интерфейса, интегрирования с другими языками программирования, в том числе и C++. Поддерживает современные операционные системы, такие как Linux, windows, macintosh.

SWI-Prolog включает в себя быстрый компилятор, профилировщик, набор библиотек и удобный интерфейс для подключения C-модулей. Он реализован для ряда UNIX-платформ, таких, как HP, IBM Linux, для NeXT, OS/2, Sun и Sparc

Пример графического интерфейса представлен на рисунке 1.5

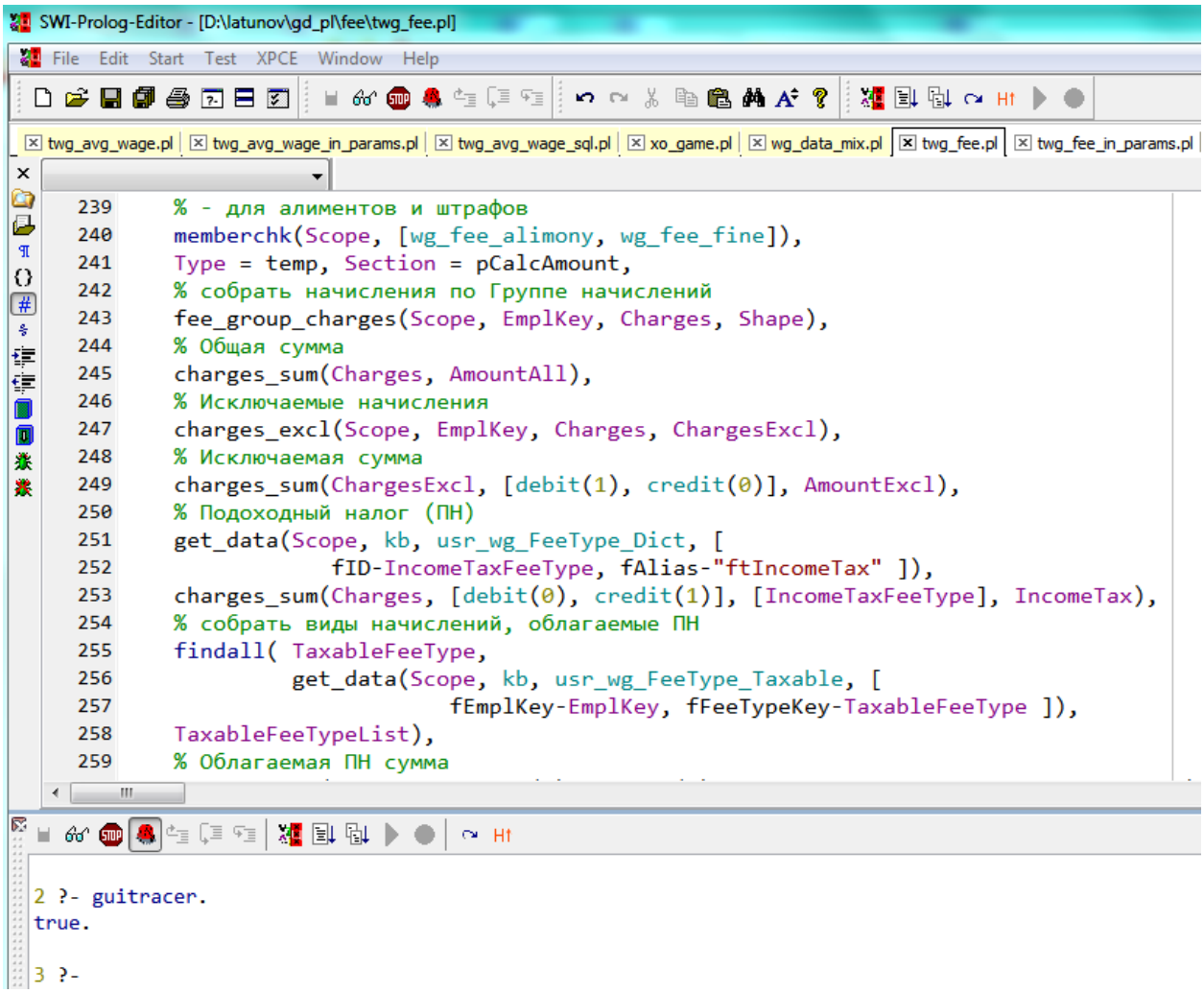

Рисунок 1.5 – Графический интерфейс SWI-prolog

1.4 Выводы

Проведя анализ предметной области и рассмотрев программные решения по заданной проблеме можно сделать выделить следующие тенденции:

- Большинство существующих методов логического вывода опираются метод резолюций, который из-за последовательного выполнения не может быть оптимально распараллелен на современных ЭВМ.
- Существует небольшое количество программных решений реализующий метод деления дизъюнктов и отсутствие программ, реализующих логический вывод в логике предикатов первого порядка.
- Проблема представления знаний имеет обширную теоретическую и практическую базу.

2 Техническое обоснование проекта. Задачи курсового проектирования

В данном разделе, на основе анализа аналогичных программных решений выдвигаются требования к функциональным характеристикам, к надежности, к графическому интерфейсу, которым должно удовлетворять приложение. Также приводится обоснование необходимости разработки и назначение программы.

2.1 Обоснование необходимости разработки

В настоящее время переход от обработки данных к обработке знаний является одним из приоритетных и перспективных направлений разработки искусственного интеллекта. Одним из главных направлений данного течения является логический вывод, который имеет хорошую базу методов.

Программы, рассмотренные в пункте 1.2, имеют существенный недостаток в виде возможности логического вывода только в логике высказываний. Модификация данных программ затруднена изменением их архитектуры. В связи с этим принято решение разработать собственную программу логического вывода в логике предикатов первого порядка методом деления дизъюнктов с использованием современных средств разработки программного обеспечения.

2.2 Постановка задачи на проектирование

Разрабатываемая программа предназначена для научного исследования логического вывода методом деления дизъюнктов в логике предикатов первого порядка. Основная цель данной программы разработка практического инструмента реализующий теоретическую базу метода деления дизъюнктов. Данная программа на целена на демонстрацию возможностей данного метода, а так же для практического применения как студентами так и преподавателями.

2.3 Функциональные характеристики

Разрабатываемая программа должна обладать следующими характеристиками:

- Реализация дедуктивного метода логического вывода в логике предикатов первого порядка.
- Обработка исходных данных, которыми являются набор фактов, высказываний, выводимых высказываний, которые представлены в виде последовательности дизъюнкций – предикатов, разделенных дизъюнкцией.
- Возможность просмотра логов процесса вывода
- Сохранение в виде файла и загрузка базы знаний и всех сопутствующих пользовательских настроек
- Реализация меню настроек, внутри которых возможно менять вид вводимых констант, изменение логики редактора высказываний.

2.4 Требования к программной совместимости

Разрабатываемое программное обеспечение должно работать на всех современных операционных систем для рабочего стола: Windows 7/8/10, Mac OS X, Ubuntu 16.

Также разрабатываемая программа должна одинаково выглядеть на всех современных операционных системах.

2.5 Требования к надежности

Надежное функционирование приложения должно быть обеспечено выполнением конечным пользователем ряда требований по эксплуатации:

- Организация стабильной работы вычислительного устройства, в данном случае – персонального компьютера.
- Обеспечение достаточных вычислительных мощностей описанных в пункте 2.6.

Со стороны разработчика приложение гарантирует корректную обработку всех входных данных, и если они не соответствуют желаемым, то оповещение об этом пользователя.

2.6 Условия эксплуатации

Минимальные системные требования, которые необходимы для корректной работы программного обеспечения:

- 32-х разрядный процессор с тактовой частотой 1 Ггц или выше
- 1 Гб ОЗУ
- 30 Мб свободного места на жестком диске

2.7 Требования к исходным кодам и языкам программирования

Программное обеспечение должно быть разработано на C++ для достижения наибольшей производительности.

2.8 Требования к программной документации

Разрабатываемое приложение будет поставляться со следующей программной документацией:

- Руководство пользователя, которое включает в себя описание всех возможностей приложения и исчерпывающую информацию по назначению каждого элемента.
- Исходный код с документацией.

2.9 Требования к интерфейсу

Интерфейс программы должен быть интуитивно понятным и наиболее простым для конечного пользователя. В процессе использование программы, пользователь должен получать необходимые сообщения об ошибках, предупреждений для того, чтобы правильно реагировать на состояние программы.

В общем случае интерфейс должен состоять и следующих элементов:

- Главное меню программы состоящее из создания нового проекта, открытия существующего, сохранения проекта, меню редактирования, внутри которого возможна смена шрифта, отмена последнего действия в редакторе высказываний и тд. Меню настроек, имеющее возможность подстроить программу под нужны пользователя. Меню помощи, которое будет содержать пользовательскую документацию.
- Окно настроек, в котором будут находится пункты меню установленные заказчиком: начало констант с прописных и строчных символов, автоподстановка истинной импликации.
- Окно редактирования высказываний, содержащее редактор высказываний, окно логов.

2.10 Выводы

Таким образом, разрабатываемое приложение должно удовлетворять следующим условиям:

- Приложение должно быть кроссплатформенным, т.е. выполняться под управлением различных операционных систем, таких как windows, linux и т.д.
- Должно иметь графический интерфейс интуитивно понятный конечному пользователю: отдельные окна ввода исходных высказываний, выводимых высказываний и результатов вывода.
- Иметь возможность настройки редактора высказываний под пользовательские предпочтения.
- Реализовать дедуктивный метод логического вывода обобщенным делением дизъюнктов

Необходимость разработки нового программного обеспечения обусловлена тем, что в ходе научных исследований в области логического вывода делением дизъюнктов были модифицированы алгоритмы процедур логического вывода, которые теперь позволяют сформировать схему логического вывода. Программа позволит более детально изучить новые методы логического вывода, выявить возможные исключительные ситуации, возникающие при решение тех или иных задач.

3 Математический аппарат и алгоритмы логического вывода

В данном разделе представлено формальное описание дедуктивного вывода методом обобщенного деления дизъюнктов в логике предикатов первого порядка. Рассмотрены процедуры унификации, частичного и полного деления дизъюнктов, согласования, а так же разработаны блок-схемы алгоритмов данных процедур.

3.1 Формальная система в логике предикатов

Логика предикатов основана на логике высказываний и имеет следующий алфавит:

1. Предметные переменные $x_1 \dots x_i$.
2. Конечное множество предметных констант $a_1 \dots a_j$.
3. Конечное множество функциональных констант $f_1^N \dots f_T^N$. (N= 1,2..)
4. Конечное множество M-местных предикатных констант A_1^M, A_2^M, …, A_K^M, …, P_K^M, … .
5. Символы исчислений высказываний: 1 (истина), 0 (ложь), v (дизъюнкция), & (конъюнкция), - (отрицание), → (логическое следование), ↔ (эквивалентность), => (символ логического вывода).
6. Символы кванторов: ∀ (всеобщности), ∃ (существования).

3.2 Операция унификации

Самой базовой процедурой метода деления дизъюнктов является унификация. Унификация необходима для приведения двух предикатов к общему виду путем подстановок.

Процедура унификации позволяет не только обнаружить одинаковые литералы, но и привести литералы к одинаковому виду. Преобразование литералов основано на подстановке термов вместо переменных.

Результатом применения процедуры унификации является подстановка вида $\lambda = \{t_1/z_1,\ t_2/z_2\ \ldots t_n/z_n\}$, где t – терм, а z – переменная, которая не содержится в терме t.

Алгоритм унификации двух литералов состоит из 7 шагов:

1. Если литералы имеют разные имена или имеют не равное количество аргументов, то унификация невозможна, иначе перейти к пункту 2.

2. Инициализировать пустое множество подстановок.

3. Если литералы не равны, то перейти к пункту 4, иначе п. 7

4. Найти первые подвыражения литералов, которые не являются одинаковыми и записать в z и t.

5. Если во множестве {z, t} z – переменная, а t – терм, который не содержит z, то добавить данную подстановку в λ, иначе унификация невозможна.

6. Выполнить подстановку λ в исходные литералы и перейти к пункту 3.

7. Унификация успешно завершена, искомая подстановка находится в λ.

3.3 Операция частичного деления дизъюктов

Частичное деление дизъюнктов реализуются путем применения процедуры образования остатков $\omega < b, d, q, n >$, где b – остаток делимое, d – остаток делитель, q – частный признак решения, который может находится в 3-х состояниях: «0» - был получен нулевой остаток, «1» - все остатки равны единице, «g» - были получены остатки, среди которых нет нулевого остатка и среди остатков есть остаток не равный 1, n – множество пар остатков делимых и делителей, где делитель получается путем применение к делителю d подстановки λ.

Основной операцией ω процедуры является нахождение матрицы «производных», которая определяется следующим образом:

$$\mu[b, d] = \frac{\partial b[L_i]}{\partial L'_k} = \Delta_{kj}$$

Результатом применения ω процедуры является множество пар остатков и делителей полученных в ходе применения процедуры.

В алгоритме ω процедуры выполняются следующие действия:

1. Составляется матрица «производных»

2. Если хотя бы одна производная имеет остаток 0, то q = «0», n = {} и переход к пункту 5, иначе к пункту 3

3. Если все производные равны 1, то q = «1», n = {}, переход к пункту 5, иначе к пункту 4

4. q = «g», n = {$<b_t, d_t>$, t = 1..T}

5. Фиксируются результаты ω процедуры.

3.4 Операция полного деления дизъюнктов

Полное деление дизъюнктов направлено на получение конечных остатков, путем последовательного применения ω процедур к исходным дизъюнктам. Конечными остатками называются остатки, которые в результате применения ω процедуры создают только остатки равные единице.

Полное деление дизъюнктов реализуется путем применения Ω процедуры. Результатом Ω процедуры является:

1. Признак решения Q, имеющий 3 значения: «1» - в результате деления посылки D на дизъюнкт d не было получено остатков, «0» - в результате деления D на дизъюнкт d был получен нулевой остаток и «G» - обозначающий наличие не нулевых и не единичных остатков.

2. Дизъюнкт посылки D.

3. Дизъюнкт заключения d.

4. Множество остатков N.

Процесс выполнения Ω процедуры состоит из 3-х шагов:

1. Подготовительный. На данном этапе производится выполнение процедуры частичного деления дизъюнктов и анализируется полученный признак q. Если q = «0», то решение найдено, если q = «1», то происходит переход к заключительному шагу. Если q = «G», то происходит переход к основному шагу.

2. Основной. При первом выполнении основного шага к дизъюнктам делителям, полученным в ходе подготовительного шага, добавляются инверсии литералов фактов. Для каждой пары остатка и делителя выполнятся ω процедура и анализируются частные признаки решения q. Если среди данного множества есть q = 0, то происходит переход к заключительному шагу. Для всех результатов ω процедуры, значение которых равно «g» производится повторное выполнение ω процедуры.

3. Заключительный. На данном этапе производится процесс поглощения остатков. Дизъюнкт b поглощается дизъюнктом e, тогда когда возможно найти такую подстановку λ, при которой $b \subseteq \lambda e$. Если поглощены все остатки, то N = {1}, G = 1.

Существует операция полного деления дизъюнктов, результат которой есть конъюнкция остатков b_1, b_2 … b_k, полученных во множестве N.

3.5 Базовая процедура вывода

Базовая процедура вывода позволяет делать шаг логического вывода, в результате которого преобразуется дизъюнкт выводимой секвенкции во множество новых дизъюнктов, которые используются в следующем шаге вывода.

Условное обозначение базовой процедуры вывода $w = < M, d, o, p, m >$, где $M = \{D_1, D_2, \dots, D_i\}$ – множество исходных дизъюнктов, $D = L_1^i \vee L_2^i \vee \dots \vee L_k^i$ – дизъюнктов состоящий из дизъюнкции литералов L, $d = L_1 \vee L_2 \vee \dots \vee L_K$ – выводимый дизъюнкт, o <c, C> - пара множеств остатков до выполнения процедуры полного деления (c) и после выполнения процедуры полного деления, p – признак окончания логического вывода, где «0» - вывод успешно завершен, «1» - вывод невозможен, «g» - необходимо продолжать вывод, m – множество новых выводимых секвенций.

Основной операцией базовой процедуры вывода является процедура полного деления.

W процедура состоит из следующего набора действий:

1. Исходные дизъюнкты при помощи Ω процедуры делятся на выводимый дизъюнкт $D_i \Omega \mathrm{d}$, в результате чего получаются конъюнкции остатков деления. Если хотя бы один из полученных остатков равен 0, то p принимается равный «0» и происходит переход к шагу 4.

Если результат выполнения всех Ω процедур равен 1 (Q = 1), то принимается p равное «1», вывод невозможен, происходит переход к пункту 4.

Формируется множество E из не схожих между собой остатков b_i. Если среди множества E окажутся остатки из множества *c*, то они исключаются. Таким образом формируется новое множество остатков *e*, которое содержит остатки не встречаемые ранее. Если множество *e* является пустым, то p = «1», происходит переход к пункту 4.

2. Формируется новое множество текущих остатков $\mathrm{C} = \mathrm{c} \cup \mathrm{e}$. Из элементов множества C формируется конъюнкция вида $\bigwedge_{i=1}^{n} b_i \to 0$, если левая часть выражения после упрощения равна 0, то p = 0, происходит переход к шагу 4.

3. Из множества *e* создается конъюнкция остатков $\bigwedge_{i=1}^{n} b_i \to 0$, которая посредством переноса остатков в правую часть образует новый выводимый дизъюнкт. Происходит переход к шагу 4.

4. Окончание w процедуры, формирование результатов.

3.6 Базовый метод дедуктивного логического вывода

Базовый метод вывода осуществляет логический вывод за конечное количество шагов последовательно применяя w процедуру на каждом шаге вывода.

Основная идея базового метода заключается в инверсии выводимого дизъюнкта, создание противоречия. Если в ходе выполнения логического вывода при помощи w процедуры был получен 0, то соответственно было

получено противоречие.

Перед началом логического вывода происходит проверка на наличие выводимых дизъюнктов среди исходных дизъюнктов-фактов. Данная проверка осуществляется при помощи операции частичного деления дизъюнктов, описанной в пункте 3.3. Если в результате деления дизъюнкта факта D на дизъюнкт вывода d получается остаток 0, то дизъюнкт d исключается из списка выводимых.

На этапе подготовки выполнения процедуры формируется множество выводимых дизъюнктов m. Далее для каждого дизъюнкта из множества m выполняется w процедура результаты которой собираются во множество P. Если P = 0, то вывод успешно завершен, если P = 1, то вывод неудачен, иначе производится повторное выполнение шага логического вывода при помощи w процедуры.

3.7 Обобщенный метод деления дизъюнктов

Основным отличием обобщенного метода вывода является нахождения всех возможных постановок, при которых остатки получились равны 0. Таким образом, все алгоритмы процедуры логического вывода немного изменяются и добавляется новая процедура – согласование решений, направленная на согласование зависимых переменных. Процедуры частичного и полного деления дизъюнктов модифицируются, в результате чего выполняют поиск всех возможных решений, а не до первого нулевого остатка, как в обычном делении дизъюнктов.

3.8 Согласование решений

В процессе логического вывода методом деления дизъюнктов может возникнуть ситуация, при которой некоторые дизъюнкты связаны, т.е. имеют общую переменную. Для решения данной проблемы используется процедура согласования.

Исходным данным для процедуры согласования является множество всех решений, при которых были получены нулевые остатки. Результатом выполнения функции есть признак «е» обозначающий успешность завершения процедуры и множество «Ф», содержащее варианты согласованных решений.

На первом шаге процедуры проверяется возможность согласования: если хотя бы одно из множеств подстановок является пустым, то согласование

невозможно. Если все множества имеют пустые подстановки, кроме одного, то результатом согласования является непустая подстановка, иначе выполняется следующий шаг.

На втором шаге происходит формирование множества вариантов согласования путем создания множества произведений множеств подстановок и формирование путем комбинаций множества подстановок

Третий шаг выполняет функцию согласования решений, в котором для каждого варианта согласования образуется объединенная подстановка и проверяется наличие одинаковых переменных. Если были найдены одинаковые переменные, то производится попытка унификации, если хотя бы для одной из переменной унификация проходит неуспешно, то согласование для данной переменной невозможно,

Последний шаг включает в себя формирование результатов процедуры согласования. Если все объединенные подстановки не удалось унифицировать, то согласование невозможно и устанавливается флаг E, иначе из объединенных подстановок, для которых унификация прошла успешна формируется множество согласованных решений и завершается процедура согласования.

3.9 Разработка программных алгоритмов

Проанализировав описание математического аппарата для задачи логического вывода методом деления дизъюнктов были выделены следующие алгоритмы: алгоритм унификации, алгоритм частичного деления дизъюнктов, алгоритм полного деления дизъюнктов, алгоритм базовой процедура вывода.

Алгоритм унификации двух литералов состоит из попарного сравнения элементов двух литералов и произведения подстановки. Граф-схема алгоритма унификации представлена на рисунке 3.1.

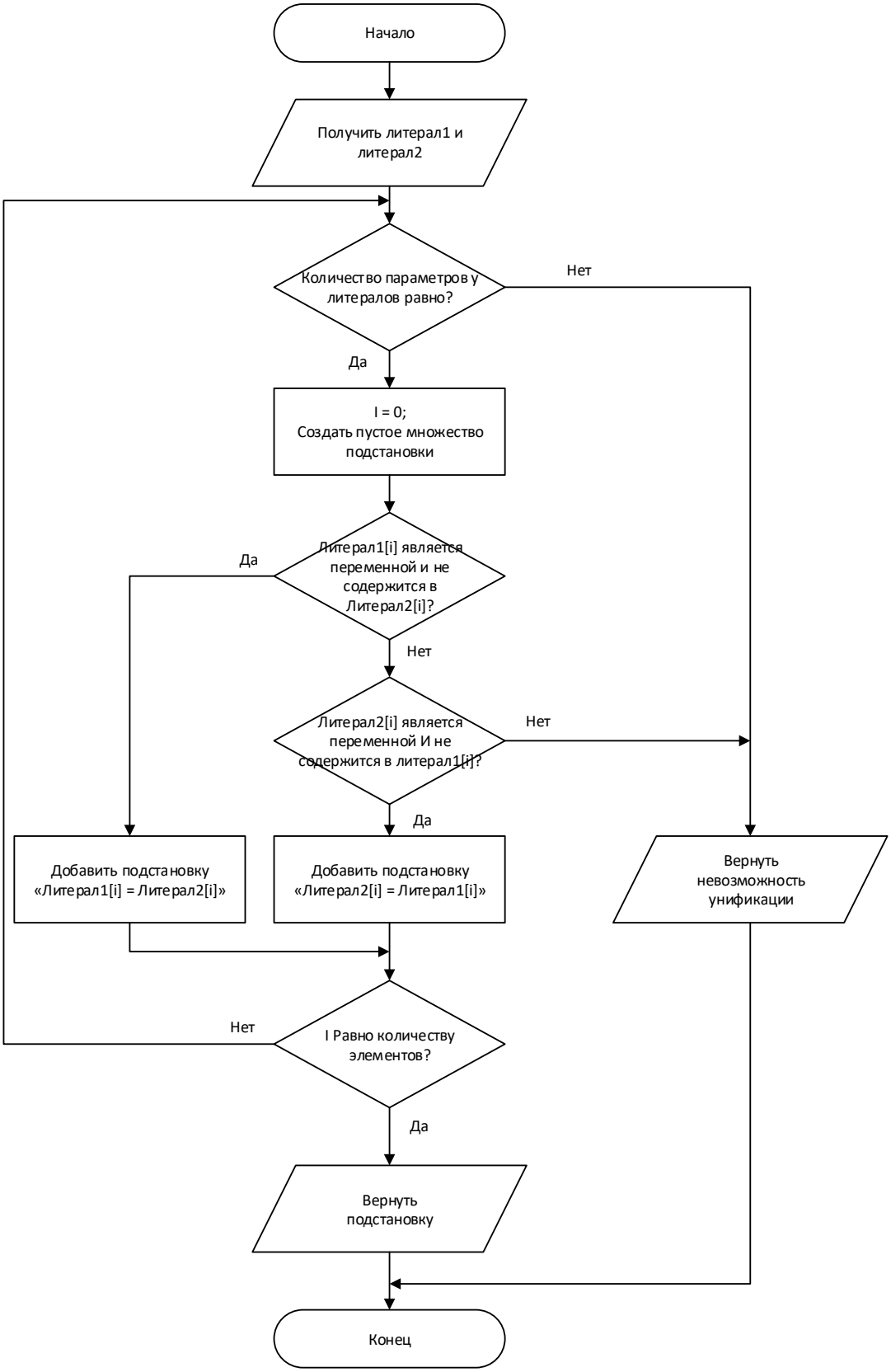

Рисунок 3.1 – Граф-схема алгоритма унификации двух литералов

Процедура модифицированного частичного деления заключается в последовательной унификации пары литералов делимого и делителя и нахождения остатка от деления. В общем случае процедура частичного деления выполняет следующие шаги:

1) Создание пустой матрицы остатков и массива структуры, состоящей из остатка и делителя.

2) Последовательный проход по всем литералам делимого делителя.

3) Произведение унификации между литералом делимого и делителя

4) Применение подстановки, полученной в ходе унификации литералов ко всему дизъюнкту и добавление полученного остатка в матрицу производных.

5) Вычеркивание из дизъюнкта делителя литерала, который унифицировался, и применение подстановки к оставшемуся дизъюнкту. Полученный дизъюнкт добавляется в результирующее множество

Граф-схема алгоритма частичного деления дизъюнктов представлена на рисунке 3.2.

Алгоритм полного деления дизъюнктов выполняет последовательное выполнение частичного деления дизъюнктов до тех пор пока признак q не останутся только нулевые или единичные остатки. При первом выполнений основного шага к дизъюнктам делителям добавляются инверсии литералов фактов.

Граф-схема алгоритма полного деления дизъюнктов представлена на рисунке 3.3

Алгоритм базовой процедуры вывода основывается на последовательном применений Ω процедуры к исходным дизъюнктам:

1. C = d. Пока не закончился список исходных дизъюнктов применять процедуры полного деления дизъюнктов.

2. Проверить полученные остатки. Если один из остатков равен 0, то принять p = 0 и перейти к шагу 8.

3. Если все остатки равны 1, то принять p = 1 и перейти к шагу 8.

4. Исключить все остатки равные 1.

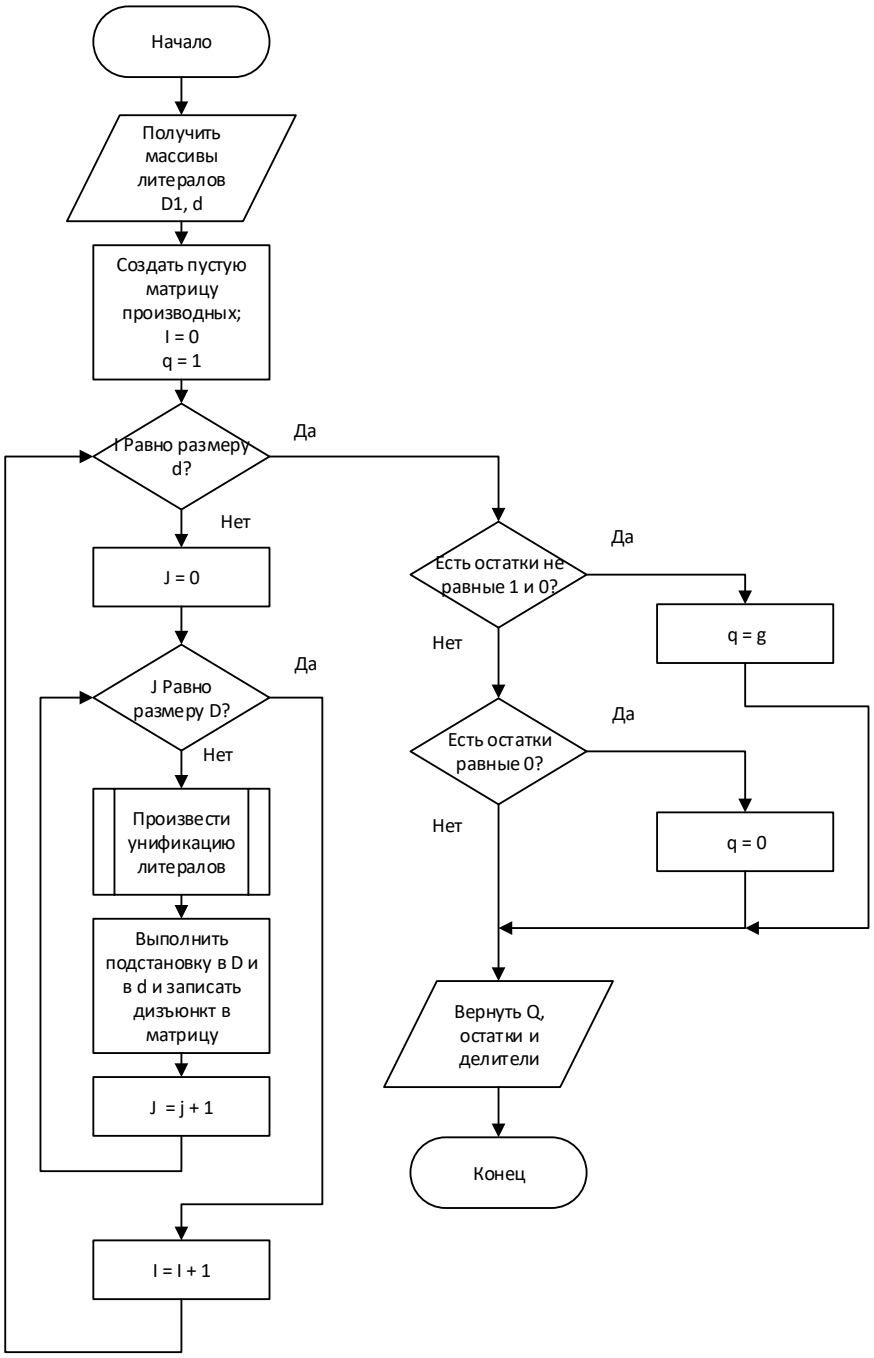

Рисунок 3.2 – Граф-схема модифицированного частичного деления дизъюнктов

5. Исключить из множества остатков, остатки, которые есть во множестве *c*.

6. Составить из оставшихся остатков конъюнкцию, если конъюнкция равна 0, то p = 0 и переход к шагу 8.

7. Сделать инверсию остатков и перейти к шагу 1.

8. Вернуть p.

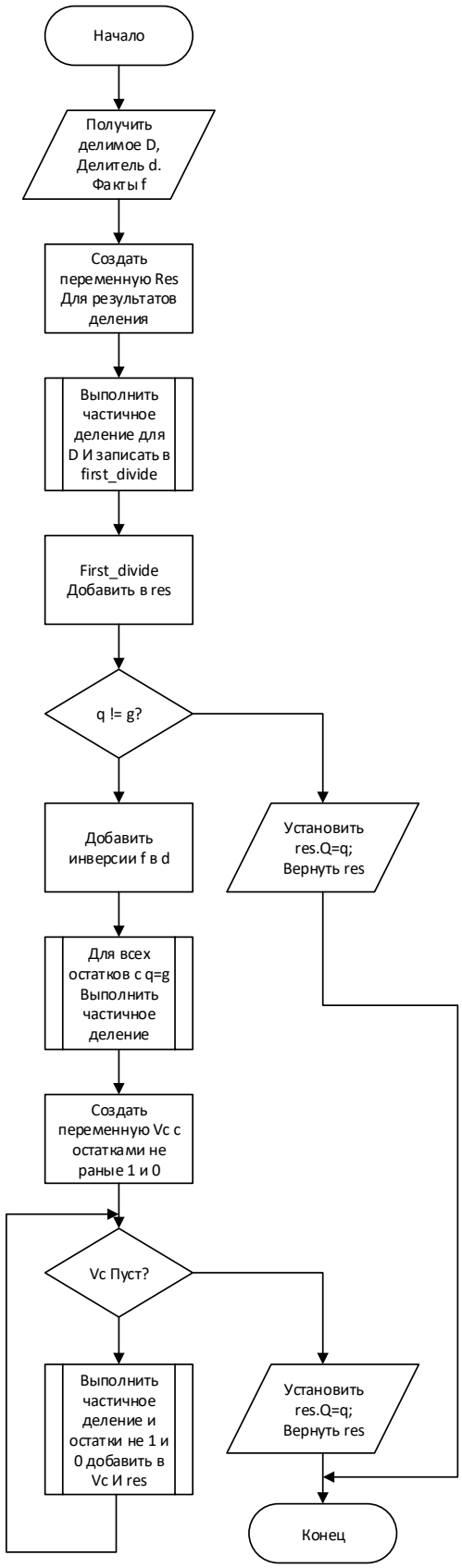

Рисунок 3.3 – граф-схема модифицированного полного деления дизъюнктов

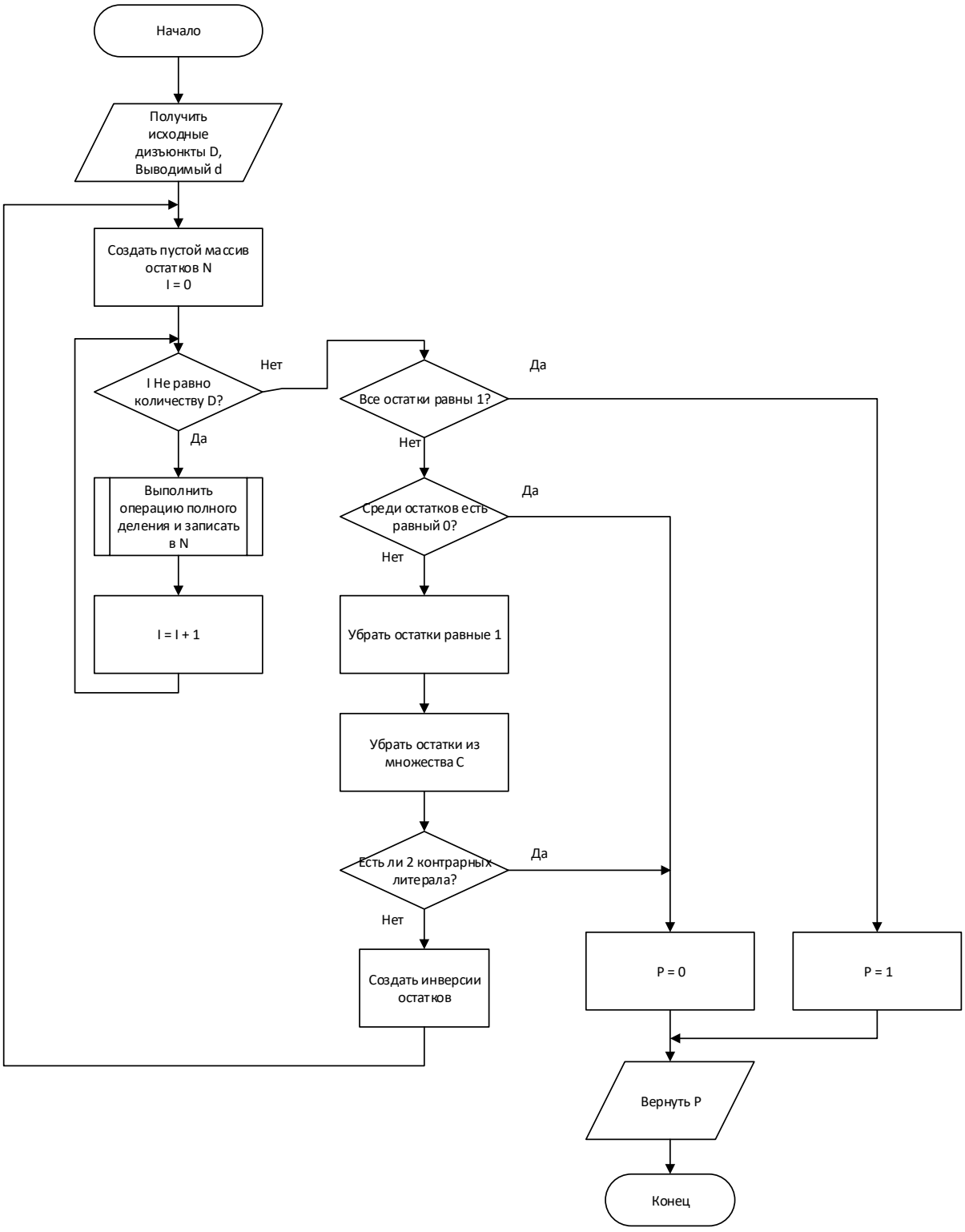

Рисунок 3.3 – граф-схема базовой процедуры вывода

3.10 Вывод

В результате курсового проектирования был рассмотрен метод логического вывода методом деления дизъюнктов. Данный метод был разбит на отдельные части: унификацию, частичное деление, полное деление и базовый дедуктивный метод. Каждая часть данного метода была рассмотрена и разбита на последовательность шагов.

Были разработаны программные реализации данных алгоритмов, которые будут составлять ключевую часть в разрабатываемой программе. Разработанные блок-схемы алгоритмов могут без труда быть реализованы в виде программного кода на любом удобном языке программирования.

4 Разработка программы дедуктивного логического вывода методом деления дизъюнктов

В данном разделе описывается процесс разработки программы, выбор инструментальных средств разработки, описание структуры программы, основных классов. Приведено детальное описание алгоритмов работы каждого из функциональных блоков программы с примерами исходного кода на языке C++.

4.1 Обобщенная структура программы

Разрабатываемая программа в рамках курсового проектирования имеет следующую структуру, представленную на рисунке 4.1.

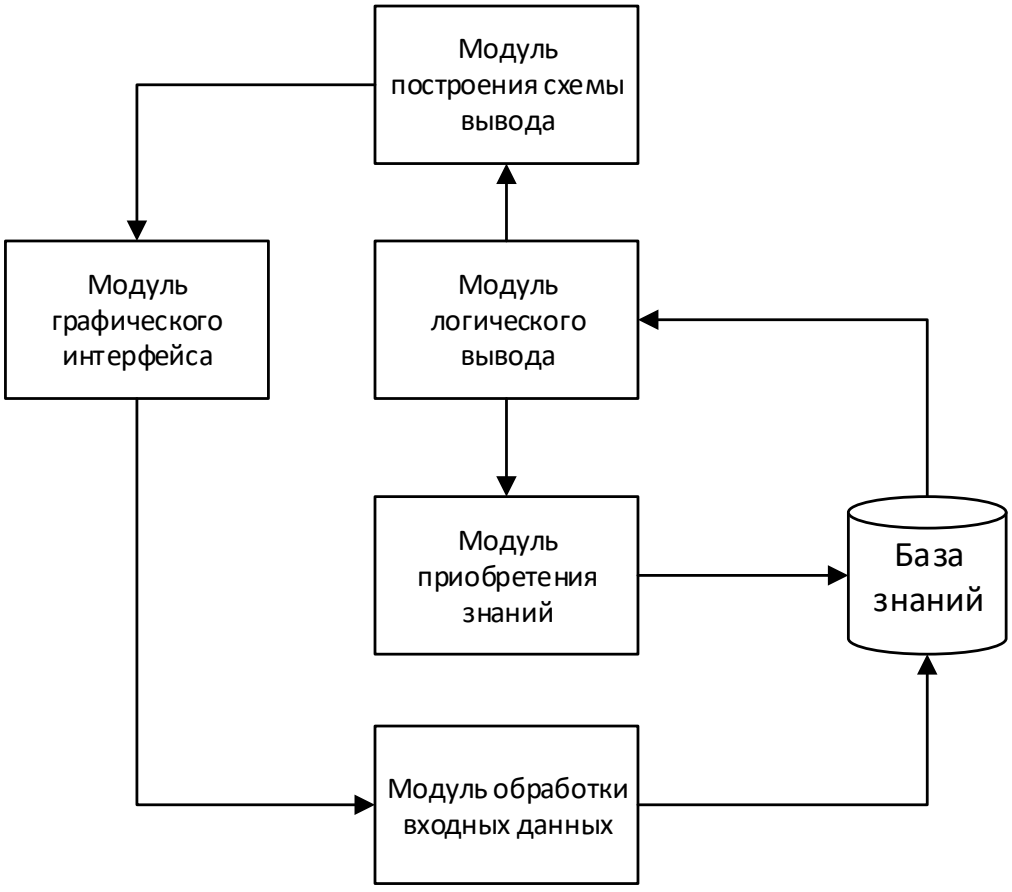

Рисунок 4.1 – Обобщенная структура программы

В модуле графического интерфейса описываются все элементы графического интерфейса: кнопки, поля ввода, меню. Так же в данном модуле находятся все обработчики событий, которые возникают при взаимодействии с пользовательским интерфейсом.

Модуль логического вывода содержит все процедуры и функции описанные в предыдущем разделе: функция унификации, частичного деления, полного деления, базовый метод вывода. Так же в данном модуле находятся описания классов, используемых в логическом выводе: предикаты, функциональных константы, переменные и константы.

В модуль лексического и синтаксического анализа описаны классы отвечающие за лексический и синтаксический анализ входных данных для дальнейшего их преобразования в исходные дизъюнкты, которые составляют текущую базу знаний.

База знаний хранит исходные дизъюнкты, необходимые в процессе вывода.

4.2 Обзор применяемых инструментов разработки

В качестве языка для разработки программного обеспечения был выбран C++. C++ является компилируемым, строго типизированным языком программирования, который поддерживает такие парадигмы программирования как объектно-ориентированные, процедурные.

Одним из основных преимуществ языка программирования C++ является его высокая скорость работы, которая достигается путем компилирования программы под определенную аппаратную платформу. Язык C++ смешивает в себе свойства высокоуровневых (ООП) и низкоуровневых языков (работа с указателями и памятью напрямую).

Язык имеет богатую стандартную библиотеку, которая включает различные структуры данных: векторы, списки, так и различные функции, например, сортировки.

На текущий момент существует множество библиотек для языка C++, позволяющие реализовать графический интерфейс.

Juce – открытый кроссплатформенная легковесный фреймворк для реализации пользовательских интерфейсов, реализованный в виде классов на языке программирования C++. Juce поддерживает все актуальные платформы: Windows, Mac OS, Linux, Android, iOS. Данный фреймворк распостраняется под свободной лицензией GPLv2. Основным недостатком Juce является малое количество графических компонентов и нестандартный внешний вид, что может ввести в заблуждение неопытного пользователя. В настоящее время данный фреймворк практически не используется.

GTK+ - кроссплатформенная библиотека элементов интерфейса, изначально разрабатываемая для графического редактора GIMP. GTK+ состоит из двух основных компонентов: GTK, который содержит набор элементов пользовательского интерфейса и GDK, отвечающий за вывод интерфейса на экран. Внешний вид отображаемых элементов может настраиваться программистом и пользователем путем замены «движка» отрисовки. За счет этого достигается разнообразие внешнего вида разрабатываемого приложение. На основе фреймворка GTK+ были разработаны такие графические окружения как GNOME, xfce. Основным недостатком данной библиотеки является сложность написания кода.

Qt – кроссплатформенный фреймворк написанный на языке C++. Данный фреймворк позволяет разрабатывать приложение, которое может быть запущенно на различных аппаратных и программных платформах без изменений или с минимальными изменениями. Qt разрабатывается одноименной компанией под свободной программной лицензией. Qt в своей основе использует стандартный C++ с расширениями в виде сигналов и слотов, которые позволяют просто обрабатывать события. Фреймворк поддерживает множество компиляторов таких как GCC и Visual Studio. Пример интерфейса написанного при помощи Qt представлен на рисунке 4.2.

QtQuick – свободно-распространяемый фреймворк разработанный в рамках Qt. Позволяет разрабатывать интерфейсы различной сложности, которые в основном разрабатываются для мобильных приложений. QtQuick включает в себя декларативный скриптовый язык QML. QtQuick позволяет писать интерфейсы не зависимые от определенного языка, таким образом возможно разрабатывать отдельно интерфейс программы и ее логику.

Интерфейс приложения, использующее QtQuick описывается декларативно при помощи QML. Синтаксис QML очень схож с JSON. Пример описания простейшего интерфейса на языке QML представлен на рисунке 4.3. Результат отображения – на рисунке 4.4.

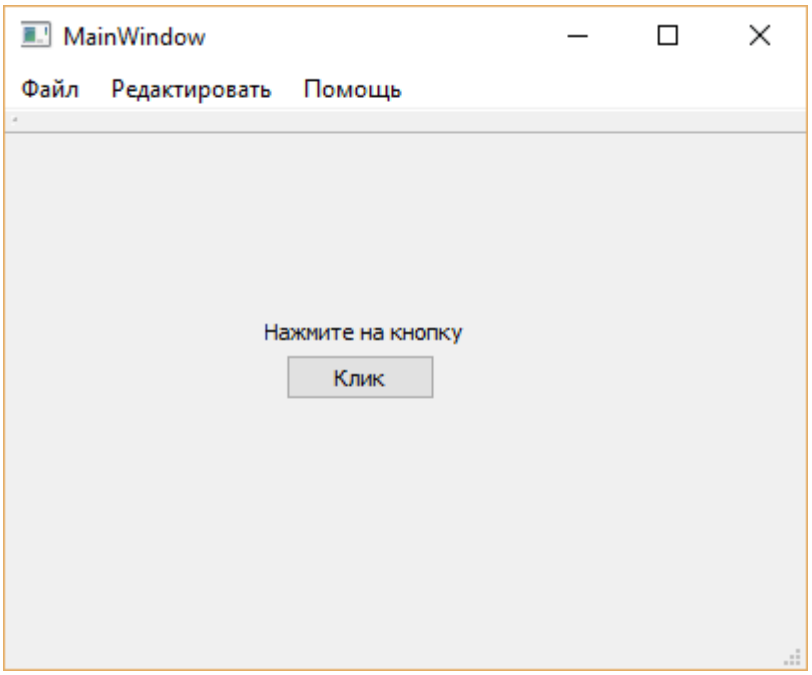

Рисунок 4.2 – интерфейс программы разработанной при помощи Qt

```
ApplicationWindow {
    visible: true
    width: 640
    height: 480

    ColumnLayout {
    anchors.centerIn: parent

        Label {
            text: "Нажмите на кнопку"
        }


        Button {
            text: "Клик"
            anchors.centerIn: parent
            Layout.margins: 10
        }

    }
}
```

Рисунок 4.3 – Исходный код простой программы

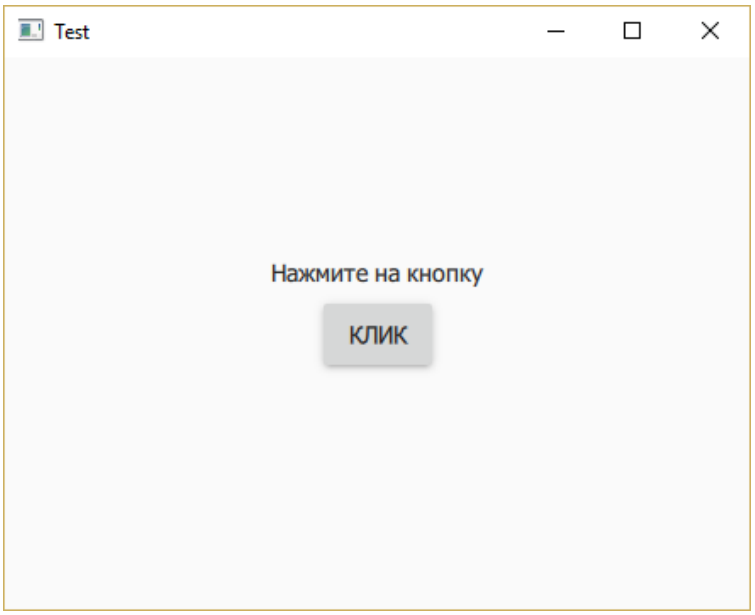

Рисунок 4.4 – Экранная форма программы на QtQuick

4.3 Разработка внутреннего представления базы знаний

В ходе анализа предметной области были выявлены основные элементы логики предикатов: переменная, константа, функциональная константа и предикат. Функциональная константа, как и предикат, является неким контейнером переменных, констант и функторов. Все эти элементы имеют общую структуру, в результате чего было решено реализовать абстрактный класс «символ», от которого наследуются все остальные элементы. Диаграмма классов Symbol, Constant, FuncConstant и Variable представлена на рисунке 4.5

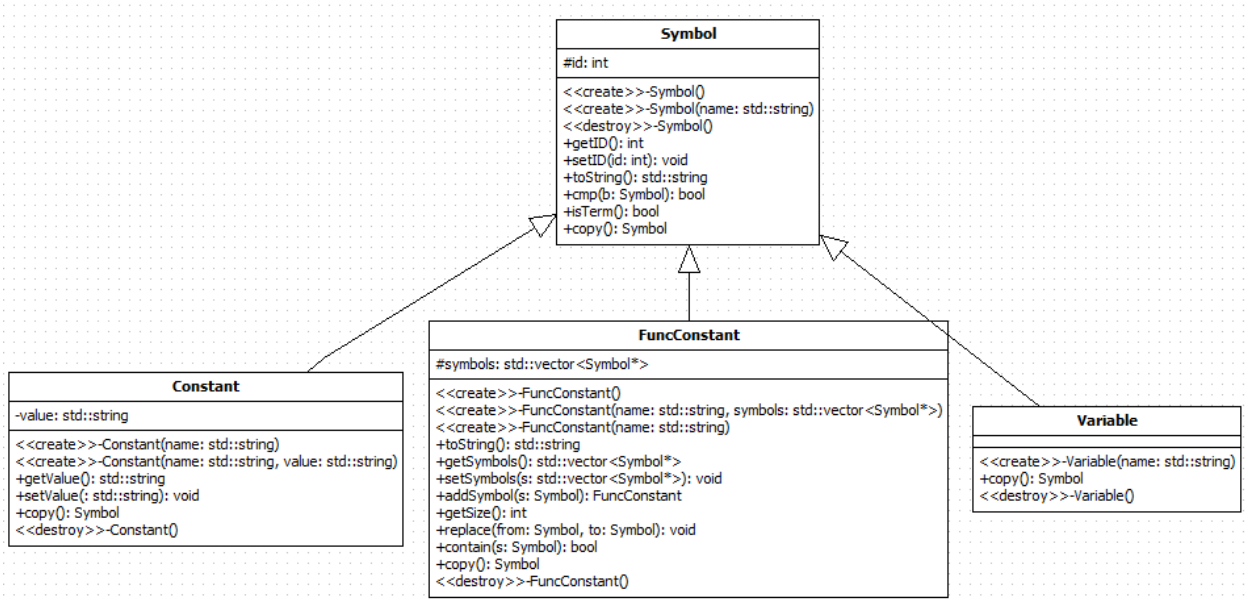

Рисунок 4.5 – Диаграмма классов Symbol, Constant, FuncConstant

Класс Symbol имеет метод cmp, который производит сравнение двух символов на эквивалентность. Если они равны, то возвращается значение истина, иначе ложь.

В классе FuncConst реализованы методы проверки наличия символа в данной функциональной константе (contain) и метод replace, который выполняет подстановку символа.

Базовым элементом дизъюнкта является предикат, который может принимать значение истина или ложь. Предикат может содержать в себе не нулевое количество символов. Из цепочки предикатов строится дизъюнкт. Дизъюнкт в программном коде может быть представлен как массив предикатов.

На рисунке 4.6 представлена диаграмма класса Predicate, в котором соответственно описывается структура предиката.

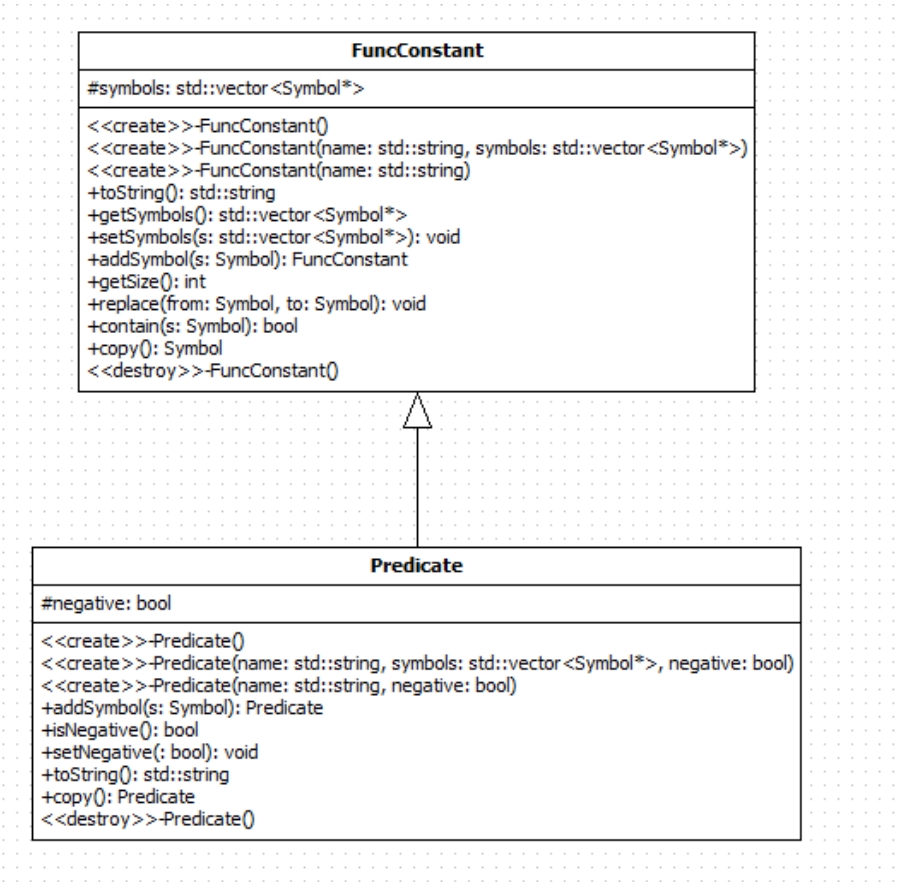

Рисунок 4.6 – Диаграмма классов Statement и Predicate.

Множество предикатов образуют высказывания На рисунке 4.8 представлен класс Statement, описывающий структуру высказывания

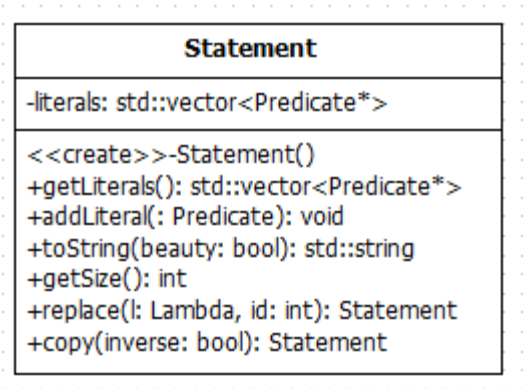

Рисунок 4.7 – Диаграмма класса Statement

Таким образом Statement является контейнером предикатом и определяет одно высказывание в базе знаний.

База знаний содержит в себе множество высказываний. Такую структуру можно представить в виде массива из высказываний. Общая структура базы знаний представлена на рисунке 4.8.

Рисунок 4.8 – Общая структура базы знаний

4.4 Основный структуры модуля логического вывода

Результатом выполнения процедуры унификации является подстановка, которая представляет из себя массив элементов замен. Диаграмма класса Lambda, который хранит элементы структуры подстановки, представлена на рисунке 4.9.

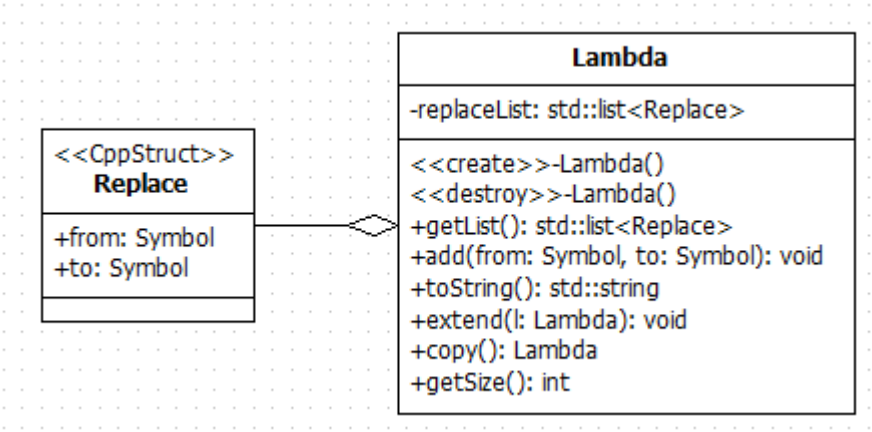

Рисунок 4.9 – Диаграмма классов Lambda и Replace

Результатом выполнения логического вывода является построение дерева вывода, в вершинах которого находятся промежуточные результаты. Основным классом является класс step, который содержит указатели на других потомков, исходные высказывания, выводимое высказывание, остатки и результаты выполнения операций полного деления, который описан в классе Omega. Диаграмма класса Step представлена на рисунке 4.10.

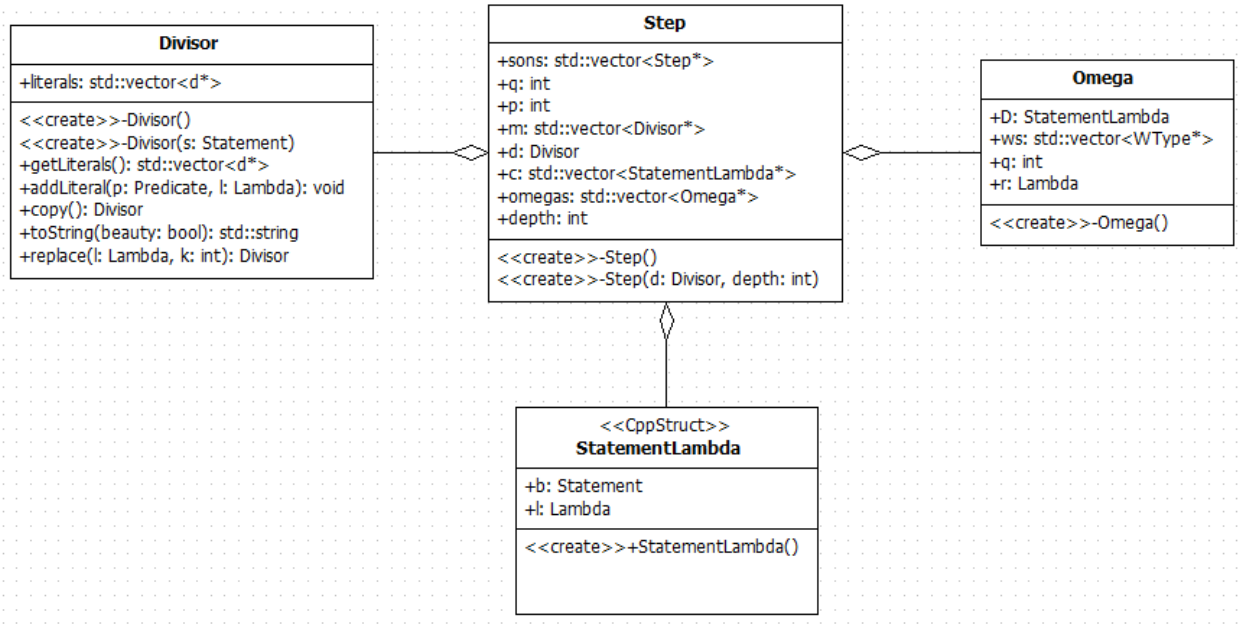

Рисунок 4.10 – Диаграмма класса Step

Класс Omega в свою очередь содержит результаты выполнения полного деления: исходные дизъюнкты, выводимый дизъюнкт, признак q, и указатель на структуру класса, содержащего результаты частичного деления, класс Wtype. Диаграмма класса представлена на рисунке 4.11

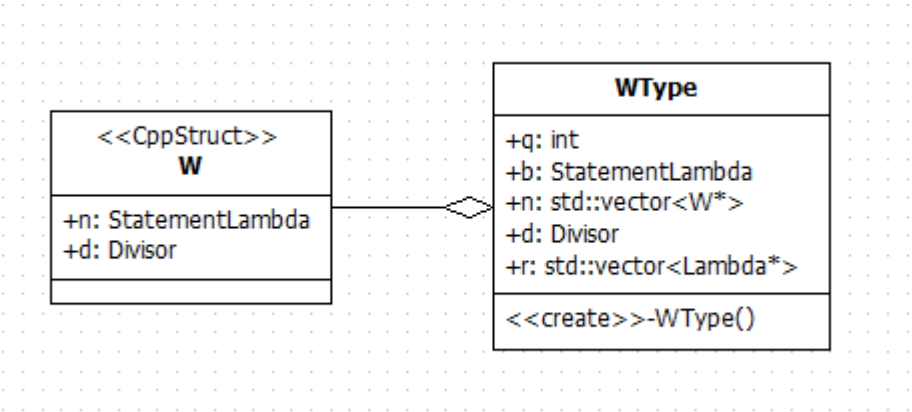

Рисунок 4.11 – Диаграмма класса Wtype

4.5 Разработка модуля логического вывода

Главной функцией модуля логического вывода является функция conclusion, на вход которой подаются исходные высказывания, выводимое высказывание, указатель на родительский узел и глубина вывода. Данная функция рекурсивно строит дерево логического вывода до тех пор пока значение глубины не будет равно ограничению.

Алгоритм состоит из следующих шагов:

1. Если глубина равно ограничению, то закончить
2. Если нет указателя на родителя, то создать корневой элемент дерева, иначе создать элемент в дереве
3. Вызвать функцию шага логического вывода
4. Для каждого нового выводимого высказывания вызвать функцию conclusion с именем родителя, равного данному узлу
5. Вернуть корень дерева.

Исходный код функции представлен на рисунке 4.11

```
Step *conclusion(std::vector<StatementLambda*> D, Divisor* d, int depth,
Step *parent, int ind)
{
    Step* root = NULL;
    if (parent == NULL) {
        root = takeStep(D, d, NULL, 1);
        root->depth = 1;
        root->ind = ind;
    }
    else {
        if (parent->depth == depth) {
            return NULL;
        }
        root = takeStep(D, d, parent, ind);
    }

    if (root == NULL) return NULL;

    int i = 1;
    for (Divisor* di : root->m) {
        if (di->getLiterals()->size() < 1) continue;
        Step* si = conclusion(D, di, depth, root, i);
        if (si != NULL)
            root->sons.push_back(si);
        si = conclusion(root->c, di, depth, root, i);
        if (si != NULL)
            root->sons.push_back(si);

        i++;
    }
    return root;
}
```

Рисунок 4.11 – Исходный код функции conclusion

Функция takeStep реализует процедуру шага логического вывода путем последовательного применения полного деления дизъюнктов к исходным высказываниям. Наибольший интерес в данной функции представляет последний шаг – упрощение выражения конъюнкции остатков. Алгоритм состоит из следующих шагов:

1. Выполняется последовательный поиск предикатов с одинаковыми идентификаторами

2. Если такая пара была найдена, то происходит проверка наличия логического отрицания

3. Если оно одинаково у обоих предикатов, то один из предикатов удаляется

4. Если знаки у предикатов разные, то данное выражение равняется 0 и вывод успешно завершается.

Функция takeDivision позволяет выполнить процедуру полного деления дизъюнктов по заданным фактам, дизъюнкту, который необходимо поделить и выводимый дизъюнкт. Данная функция в своей основе использует цикл, который выполняется до тех пор, пока есть не 1 и не 0 остатки.

Функция part_divide выполняет частичное деление. Результатом данной фунции является объект типа WType, содержащий в себе остатки деления и подстановки используемые при делении высказывания на выводимый дизъюнкт.

Исходный код реализации модуля логического вывода представлен в репозитории gitHub пользователя nellrun проекта IP.

4.6 Разработка модуля обработки входных данных

В ходе разработки программы логического вывода требовалось разбирать исходные данные, которые вводит пользователь, а именно исходные и выводимые дизъюнкты. По заданному техническому заданию, на вход программы должны подаваться дизъюнкты приведенные к сколемовскому виду, например: $1 \rightarrow P(a,b)\ v\ \bar{P}(f(b,c),a)$.

Заданная грамматика является контекстно-свободной по иерархии Хомского. Данная грамматика описывается при помощи формы Бэкуса-Наура и должна содержать следующие элементы:

- Множество терминальных символов, являющиеся основой заданной грамматики.

– Множество нетерминальных символов, состоящих из терминалов.
– Множество правил, описывающих заданную грамматику
– Нетерминальный символ, являющийся началом для грамматики

Форма Бэкуса-Наура разрабатываемой грамматики представлена на рисунке 4.11

```
<program> ::= <statement>  | <statement> <statement>.
<statement> ::= <predicate> <or> <statement> | <predicate> ".".
<predicate> ::= <predicate-id> <paren-expr> | <not> <predicate-id> <paren-
expr>.
<paren-expr> ::= "(" <expr> ")".
<expr> ::= <term> | <term> "," <expr>.
<term> ::= <var> | <const> | <func> <paren-expr>.
<not> ::= "not" | "!" | "-" | "~".
<or> ::= "v" | "or" | "|".
<var> ::= "a" | "b" | ... | "z" | "aA" | "aa" | ... | "zzzzzzzzzzzzzz".
<const> ::= "A" | "B" | ... | "Z" | "AA" | "Aa" | ... | "Zzzzzzzzzzzzzz".
<func> ::= "A" | "a" | "B" | "b" | ... | "z" | ... | "aA" | "aa" | "AA" | "Aa" | ... |
"Zzzzzzz".
<predicate-id> ::= "A" | "a" | "B" | "b" | ... | "z" | ... | "aA" | "aa" | "AA" | "Aa" | ...
| "Zzzzzzz".
```

Рисунок 4.11 БНФ для заданной грамматики языка

Начальным символов в данной грамматике является «Program», который в свою очередь содержит одно или несколько высказываний (statement).

Высказывание состоит в свою очередь из одного или нескольких предикатов, если предикатов несколько, то они должны разделяться нетерминальным символов логического или. Обязательным признаком окончания высказывания является символ точки.

Для наибольшего удобства нетерминальные символы логических операций таких как отрицание или дизъюнкция представлены в различных вариантах, которые наиболее часто используются как в языках программирования так и в литературе.

Для обработки исходных данных было решено разработать лексический анализатор по заданной грамматике, в задачи которого входило распознавать

во входной последовательности так называемые «токены» - лексическая единица заданной грамматики.

Лексический анализатор представляет из себя конечный автомат, который распознает последовательность символов. Существует множество программных средств реализующих автоматическую генерацию программного кода лексического анализатора по заданным токенам. Одной из таких программ является Coco/R, позволяющую по заданным входным символам, токенам сгенерировать программный код лексического анализатора для таких языков программирования как C++, C#, Java и других. Описанные в файле с расширением .atg символы и токены подаются на вход в программу, которая на выходе выдает готовый программный код. Диаграмма класса Lexer, который описывает лексический анализатор, показана на рисунке 4.12.

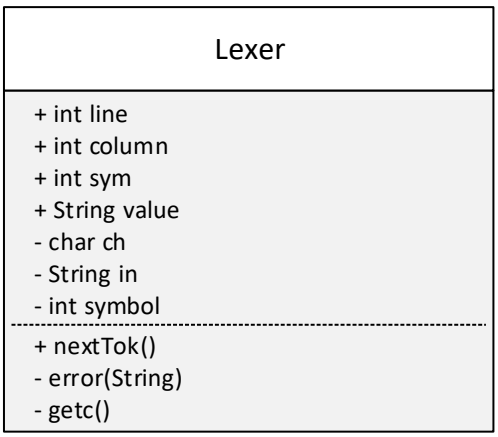

Рисунок 4.12 – Диаграмма класса Lexer

Данный класс имеет два вида конструктора, позволяющие менять источник входных данных для лексического анализатора. Конструктор по умолчанию использует в качестве источника входных данных поток ввода std::in, существует так же конструктор, который получает на вход строку, которую необходимо разбить на токены. Данный класс имеет всего один публичный метод nextTok(), который выполняет чтение строки до следующего токена, тип которого записывается в поле sym, а значение в поле value.

Для определения соответствия входных данных заданной грамматики и создание элементов базы знаний необходимо разработать синтаксический анализатор, который по заданной БНФ будет строить дерево грамматического разбора.

Существует несколько вариантов построения дерева: нисходящий, предполагающий построение дерева от нетерминального начального символа

грамматики к терминалам и восходящий, который строит из терминальных символов нетерминальные по направлению к вершине дерева.

Так как заданная грамматика языка является однозначной было выбрано построение дерева сверху-внизу при помощи рекурсивного метода. Программная реализация нисходящего дерева состоит из реализации процедур реализующие одно из конструкций лексем.

Для достижения наибольшей эффективности синтаксический и лексический анализаторы работают параллельно. Синтаксический анализатор в процессе своей работы не тратит ресурсы на построение дерева разбора, а сразу создает структуру массива высказываний.

Диаграмма класса Parser представлена на рисунке 4.13

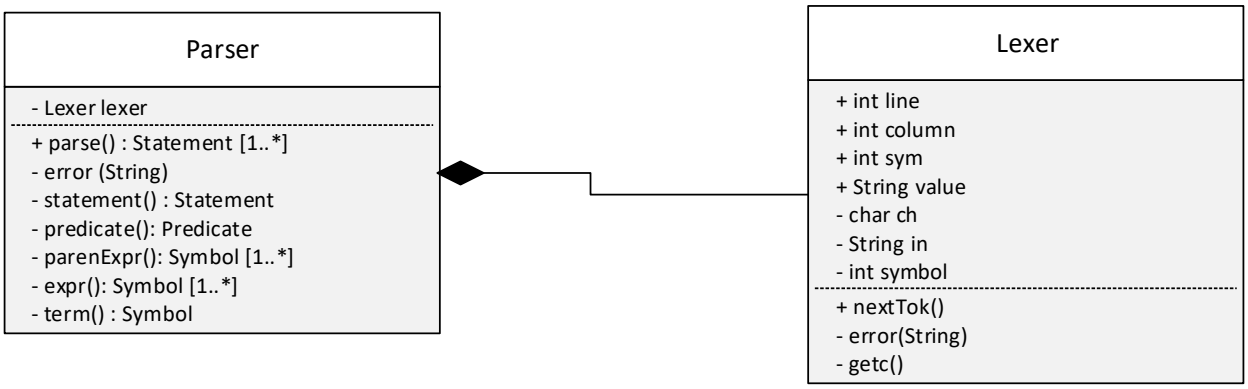

Рисунок 4.13 – Диаграмма класса Parser

Данный класс имеет один публичный метод Parse, который возвращает распознанную цепочку дизъюнктов представленной в виде структуры вектора.

Общий принцип работы синтаксического анализатора заключается в следующем. Создается экземпляр анализатора, в зависимости от выбранного конструктора парсер будет проводит анализ либо потока ввода std::in, либо переданную строку. При вызове метода parse(), лексический анализ перейдет к первому токену и будет вызвана процедура statement(), которая выполняет распознавание нетерминального символа «statement». В свою очередь внутри метода statement будет производится вызов метода predicate до тех пор, пока не будет достигнут терминальный символ окончания высказывания. Таким образом при помощи рекурсивного вызова будет обработана вся входная последовательность символов, результатом которой будет векторной представления дизъюнктов.

Пример построения дерева разбора по заданной грамматике представлен на рисунке 4.14.

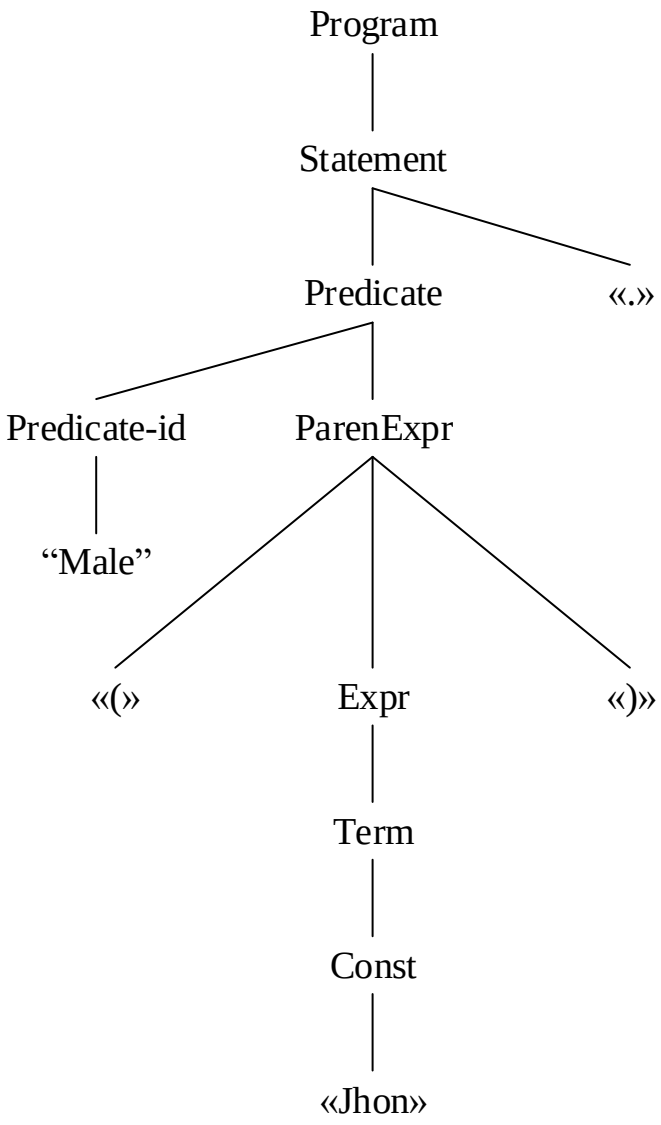

Рисунок 4.14 – Пример построения дерева синтаксического разбора

4.7 Разработка пользовательского интерфейса

Наиболее удобным инструментом для разработки пользовательского интерфейса оказался декларативный язык QML, позволяющий реализовать интерфейсы любой сложности. В последних версиях фреймворка Qt появилось возможность использования готовых тем компонентов, таких как Material, разработанной в стенах Google и Universal, разработанная Microsoft. В качестве основной темы для разработки была выбрана светлая тема Material.

Описание интерфейса программы производится в рамках операционной системы семейства Windows, но благодаря Qt на других платформах интерфейс будет выглядеть точно так же.

При запуске программы открывается базовое окно, предлагающее либо открыть существующий проект базы знаний, либо создать новый. Данное окно продемонстрировано на рисунке 4.15.

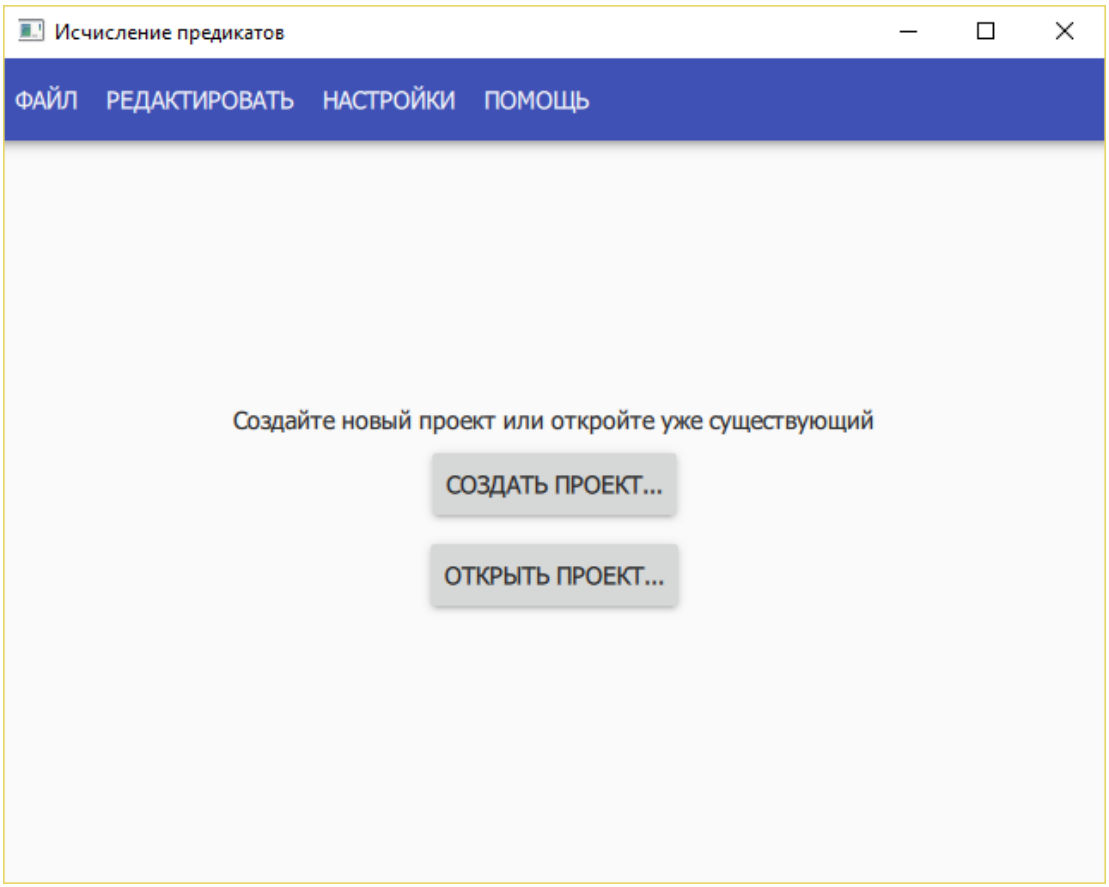

Рисунок 4.15 – Приветственное окно программы

В верхней части окна находится блок меню, который состоит из четырех разделов:

1. Раздел «Файл» содержит в себе все операции связанные с созданием, открытием и сохранением.

2. Раздел «Редактировать» позволяет менять шрифт вводимого текста, а так же выполнять операции копирования, вставки и отмены действия, если при вводе произошла ошибка

3. Элемент настройки открывает окно настроек. В данном окне появляется возможность определить формат входных данных, настроить текстовый редактор под себя. Окно настроек представлено на рисунке 4.16

4. Раздел помощи содержит информации о разработчике программы и описание основных элементов программы

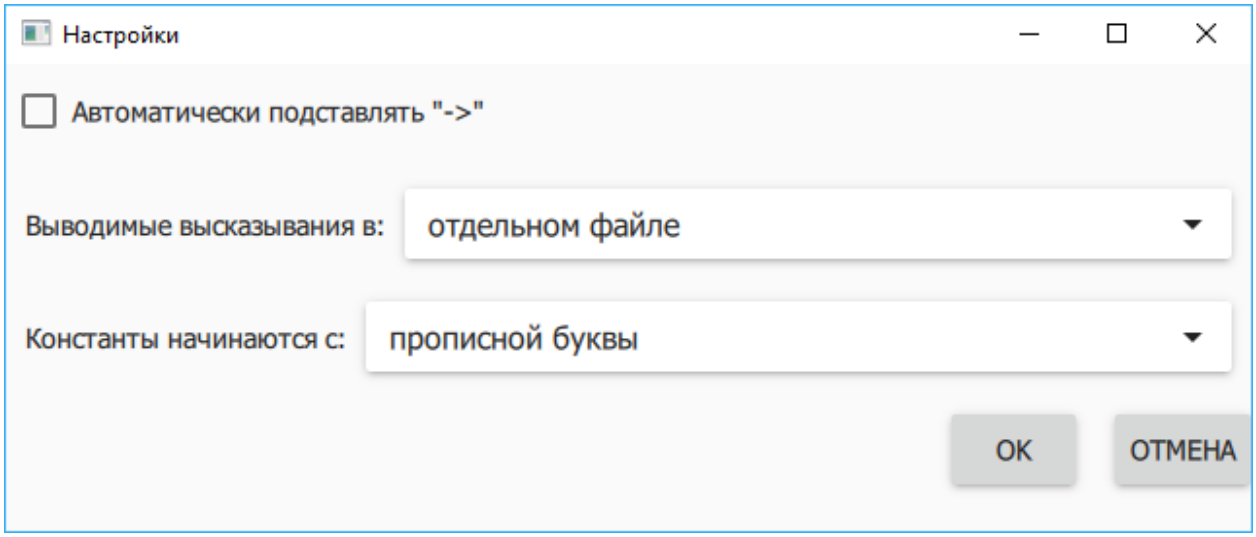

Рисунок 4.16 – Окно пользовательских настроек

После нажатия в главном окне кнопки создать или открыть проект, откроется основное окно программы, в котором возможно редактирование исходных дизъюнктов и произведение логического вывода. Главное окно программы показано на рисунке 4.17.

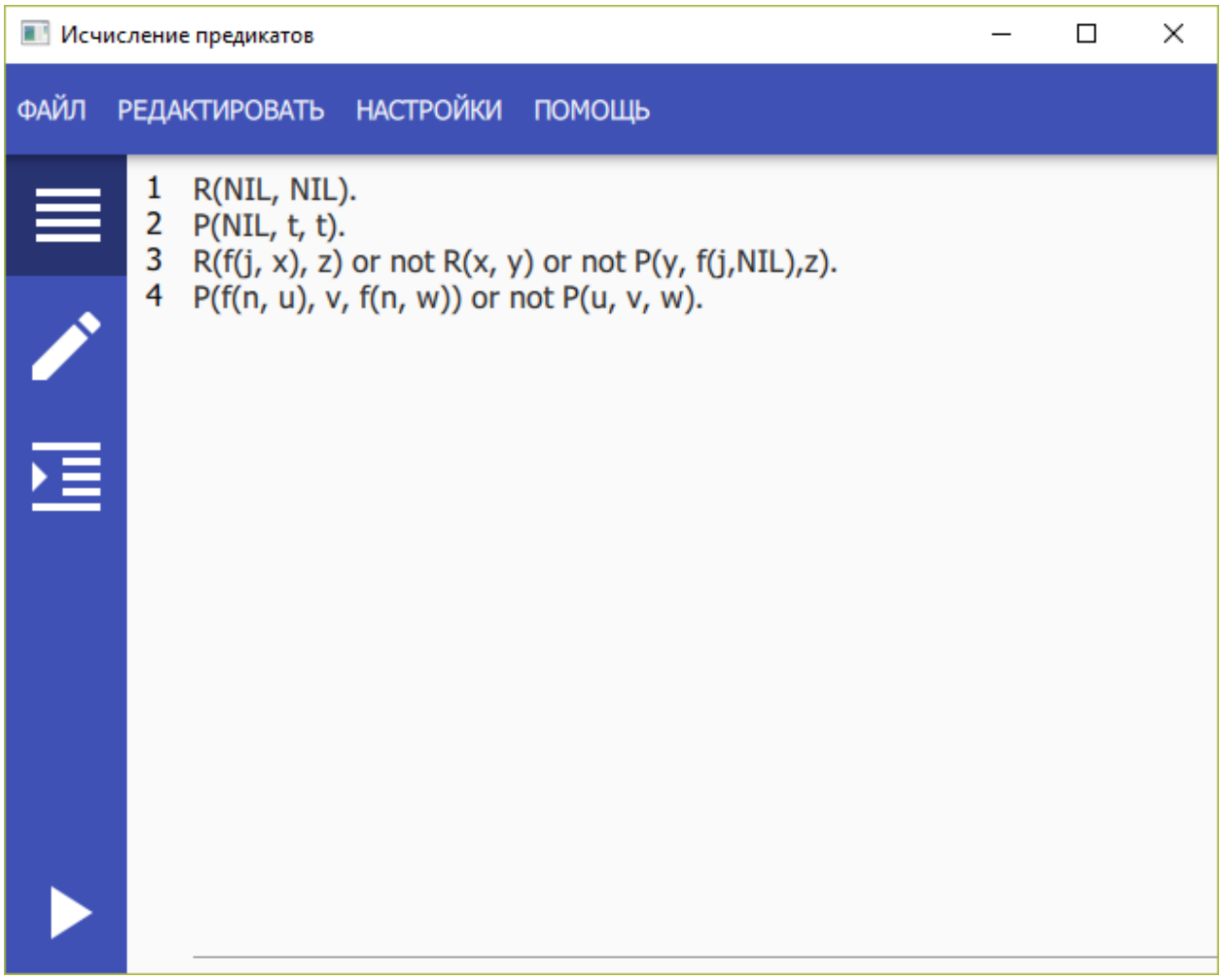

Рисунок 4.17 – Окно ввода исходных дизъюкнтов

Главное окно программы состоит из поля для ввода исходных дизъюнктов, поля для ввода выводимого дизъюнкта и кнопки запуска логического вывода.

Окно вывода логов вывода содержит в себе исходные дизъюнкты, выводимые дизънкты и последовательность шагов логического вывода: результаты процедур полного деления, частичного деления и подстановки, так же содержатся результирующие подстановки, полученные в результате применения процедуры согласования. Окно логов представлено на рисунке 4.18.

Рисунок 4.18 – Окно логов вывода

4.8 Вывод

В результате разработки программы был реализован модуль логического вывода, включающий в себя классы базовых элементов логики предикатов, а также функции необходимые для логического вывода. Данный модуль имеет слабую связь с остальными, поэтому он с легкостью может быть перенесет на другой проект.

Для обработки входных данных были разработаны лексический и синтаксические анализаторы, которые работают параллельно и без построения дерева разбора создают структуру в виде массива дизъюнктов, которые составляют базу знаний.

Для приема входных данных и отображения результатов был разработан графический интерфейс при помощи фреймворка Qt, благодаря чему удалось создать интерфейс, который будет выглядеть одинаково на всех основных платформах.

Заключение

В результате разработки программного обеспечения для осуществления логического вывода в логике предикатов первого порядка методом деления дизъюнктов было получено приложение, удовлетворяющее следующим требованиям:

- Осуществление дедуктивного логического вывода в логике предикатов первого порядка методом деления дизъюнктов
- Обработка исходных данных, которыми являются набор фактов, высказываний, выводимых высказываний, которые представлены в виде последовательности дизъюнкций – предикатов, разделенных дизъюнкцией.
- Возможность просмотра логов процесса вывода
- Сохранение в виде файла и загрузка базы знаний и всех сопутствующих пользовательских настроек
- Реализация меню настроек, внутри которых возможно менять вид вводимых констант, изменение логики редактора высказываний.

Дальнейшее развитие данного приложение будет включать в себя возможностью вводить исходные дизъюнкты в произвольного вида, а не только приведенной к сколемовскому виду. Построение схемы логического вывода, которая наглядно отображает процесс вывода.

Библиографический список

1. Ростовцев В.С. Оформление курсовых и дипломных проектов для студентов специальности 230101 [Текст] / В.С. Ростовцев, С.Д. Блинова. – Киров: Изд-во ВятГТУ, 2006. – 39 с.
2. Браудэ Э. Технология разработки программного обеспечения. – СПБ.: Питер, 2004.- 655 с.: ил.
3. Страбыкин Д. А. Логический вывод в системах обработки знаний. – СПБГЭТУ. СПБ., 1998.- 164 с.: ил.
4. Страбыкин Д. А. Формальное описание схем логического вывода в исчислении предикатов – ВятГУ. Киров., 2016. - С.2114-2118.
5. Страбыкин Д. А. Модель параллельных вычислений для логического вывода методом деления дизъюнктов – ВятГУ, Киров., 2011
6. Мельцов В. Ю. Инструментальная система для автоматизации обработки коммерческих предложений / В. Ю. Мельцов, М. Н. Земцова – 2002. – С. 10-11