

Введение

Одним из важных направлений исследования искусственного интеллекта является обработка знаний. Под знаниями подразумевается некоторая совокупность информации и правил вывода, отражающих те или иные свойства различных объектов или закономерности каких-либо процессов. Основными вопросами для изучения в данном направлении являются представление и получение знаний, а также моделирование рассуждений.

На сегодняшний день предложено множество различных способов представления знаний в памяти ЭВМ, а также методов их обработки. Все они являются попыткой отразить то, как человеческий мозг структурирует и хранит информацию и как происходят различные мыслительные процессы. Данные способы можно разделить на две большие группы.

Для первой группы характерен подход, основанный на моделировании работы множества структурно-функциональных единиц человеческого мозга – нейронов. К настоящему времени искусственные нейросети находят применение при решении множества задач, включая различные задачи кластеризации, адаптивного управления, распознавания образов и прогнозирования. Нейросеть отличается высокой скоростью получения результата (за исключением, возможно, некоторых сетей с обратной связью) и способностью к самообучению. Однако, характерным недостатком нейросетевого подхода является сложное внутреннее представление, основанное на алгебраических функциях, и, как следствие, невозможность представления семантической картины исследуемой задачи.

Вторая группа объединяет способы представления и обработки знаний в основе которых лежит моделирование рассуждений. Данный подход отражает способность человеческого мозга обрабатывать знания по определенным правилам (например, по законам логики), обобщать их и выявлять зависимости. К примеру, рассуждения, записанные с помощью логики высказываний или логики предикатов, представляются для человека достаточно естественным образом и в то же время могут быть обработаны машиной. Существенным недостатком моделирования рассуждений с помощью ЭВМ является низкая эффективность многих методов логического вывода. Переборный характер процедур вывода влечет за собой неприемлемо долгий процесс обработки большого количества логических правил.

Таким образом, актуальной задачей в области обработки знаний на сегодняшний день является исследование новых методов логического вывода, позволяющих не только с приемлемой скоростью получить результат, но и тем или иным образом отобразить нить рассуждений для дальнейшего анализа человеком. Программная реализация данных методов позволит более детально их изучить и рассмотреть возможности их дальнейшего развития.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прадл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 1 |

1 Анализ предметной области

В данном разделе рассматриваются вопросы, связанные с анализом предметной области, проводится обзор существующих решений поставленной задачи, выделяются требования к разрабатываемому программному продукту.

1.1 Обзор научно-технической литературы, анализ состояния проблемы

Методы, основанные на логическом моделировании, позволяют использовать аппарат математической логики, который к настоящему времени имеет хорошо изученную теоретическую базу и является одним из основных компонентов формальной логики. При этом подходе хранимые знания могут ограничиваться лишь некоторым множеством аксиом, а все остальные знания могут быть получены с помощью моделирования рассуждений процедурами логического вывода.

Наибольший интерес из видов логического вывода представляет дедукция и абдукция. Дедукция – это аналитический процесс вывода, при котором частные положения выводятся с помощью применения общих правил. Абдукция также является аналитическим процессом и позволяет получить объяснения для имеющихся фактов, другими словами вывести дополнительные предпосылки, исходя из исходных правил и результата. Оба вида логического вывода можно рассмотреть на примере одного силлогизма, представленного на рисунке 1.1.

Barbara

- 1) Все животные смертны.
- 2) Все люди – животные.
- 3) Все люди смертны.

Рисунок 1.1 – Силлогизм «Barbara»

Дедукцией называется логический вывод, заключающийся в выведении цели 3 из исходных посылок 1 и 2. Абдукцией же в данном случае является логический вывод, при котором выносится предположение о присутствии правила 2, при имеющемся правиле 1 и заключении 3. Для реального мира абдукция позволяет находить объяснения для наблюдаемых явлений. Однако случается, что найденные объяснения могут противоречить исходным аксиомам предметной области. По мнению Ч. Пирса[6] именно с помощью абдукции человек выдвигает различные гипотезы, которые способствуют научным открытиям.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 2 |

Множество методов логического вывода основаны на принципе резолюции. Данный принцип заключается в порождении из исходных правил новых промежуточных резольвент в соответствии с принципом суперпозиции. Но такой подход имеет существенный недостаток – большое количество промежуточных резольвент и, как следствие, замедление получения конечного результата вывода. Подобный принцип лежит и в основе популярного языка логического программирования Пролог.

Из более новых методов логического вывода следует выделить метод деления дизъюнктов[1]. Данный метод использует стратегию вывода «сначала вширь», при котором дерево вывода порождается уровень за уровнем, таким образом возможно параллельное выполнение процедур деления дизъюнктов, что существенно уменьшает общее время вывода.

Для визуализации процесса логического вывода используется специальная схема – ориентированный граф, вершинам которого соответствуют секвенции посылок, при этом входящие дуги помечены литералами из левых, а исходящие – литералами из правых частей секвенции; вершины соединены между собой в соответствии с суперпозицией их секвенций при выводе. Пример такой схемы представлен на рисунке 1.2. Представив схему вывода как набор некоторых состояний рассматриваемого объекта, можно, к примеру, осуществить прогноз его следующих состояний на основе текущего, в зависимости от условий внешней среды и наличия некоторых управляющих сигналов.

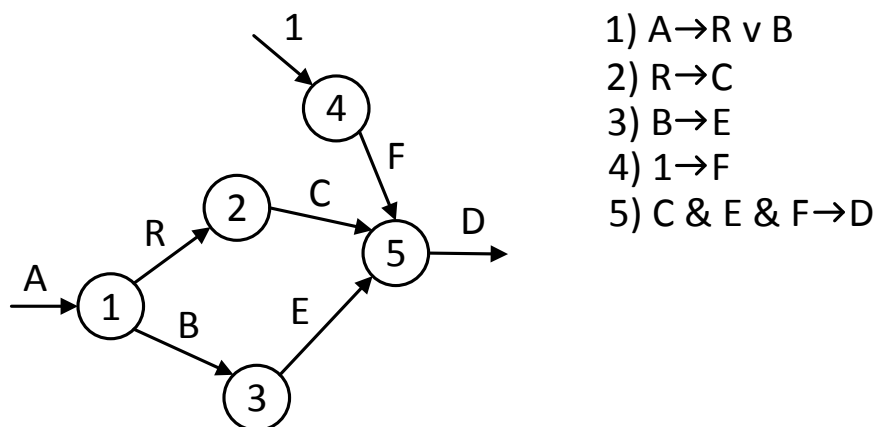


Рисунок 1.2 – Схема логического вывода заключения $A \rightarrow D$

1.2 Обзор методов и средств решения задачи

Для поддержки исследований в области обработки знаний необходима разработка специализированных программ, позволяющих автоматизировать некоторые рутинные процессы в решении определенных задач. На данный момент

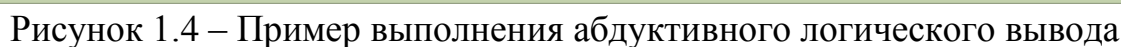
| | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | |
| | | | | | Лист | | | | |
| | | | | | 3 | | | | |

| Инв. № прокл. | Подпись и дата. | Взам. инв № | Инв. № дудл. | Подпись и дата |
|---------------|-----------------|-------------|--------------|----------------|
| | | | | |



- обработка исходных данных, представленных в виде набора правил исследуемой задачи и описанных с помощью языка декларативного программирования, разработанного в рамках данной научной работы;
- преобразование исходных формул посылок и заключения в вид дизъюнктов, которые затем обрабатываются машиной логического вывода;
- программная реализация дедуктивного и абдуктивного логического вывода методов деления дизъюнктов, а также их некоторых модификаций;
- вывод информации о ходе логического вывода с подробным описанием выполненных шагов.

| | | | | |
|---------------|-----------------|-------------|--------------|----------------|
| Инв. № продл. | Подпись и дата. | Взам. инв № | Инв. № дубл. | Подпись и дата |
| | | | | |



- отсутствие возможности редактирования базы знаний и заключений из интерфейса программы, работа ограничивается единовременной загрузкой данных из текстового файла;

- В то время как вышеперечисленные неудобства могут быть устранены в течение нескольких итераций, разработка механизма построения схемы логического вывода требует внесения существенных модификаций в программные алгоритмы и структуры данных. Все это приводит к необходимости разработки нового программного обеспечения, в основу которого лягут модифицированные методы логического вывода делением дизъюнктов с

построением схемы, используя при этом новые возможности современных инструментов разработки прикладных программ.

Выводы

Проведя анализ предметной области можно выделить следующие тенденции:

- проблема представления знаний на сегодняшний день имеет обширную теоретическую и практическую базу, предложено и разработано множество способов хранения и обработки знаний, таких как семантические сети, фреймы, различные языки и нотации, которые позволяют организовать базу знаний, готовую к обработке машиной;

- большинство программных решений в области обработки знаний опираются на метод резолюций, а также на его различные модификации, который в свою очередь обладает слабыми возможностями распараллеливания поиска решений;

- малое количество информации о практических исследованиях в области параллельных методов логического вывода, таких как логический вывод делением дизъюнктов, позволяющих использовать весь потенциал современных многоядерных и многопроцессорных вычислительных машин.

| | | | | | | | | | | |
|-----------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прот. д. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 6 |

2 Техническое и социально-экономическое обоснование проекта. Задачи дипломного проектирования

В данном разделе приводится обоснование необходимости разработки и назначение программы, формулируется основная цель разработки, а также представлена постановка задачи на проектирование, включающая в себя требования к функциональным характеристикам, к информационной и программной совместимости и к интерфейсу программы.

2.1 Обоснование необходимости разработки

В свете последних опубликованных научных работ и статей в области логического вывода делением дизъюнктов, схема логического вывода стала неотъемлемой частью решения задач, были модифицированы алгоритмы методов. Программа, рассмотренная в разделе 1.2, не имеет возможности построить схему логического вывода, а также имеет ряд небольших недостатков. Добавление механизма построения схемы в программу повлечет за собой существенную переработку программных алгоритмов и структур данных. Также, графическое отображение схемы потребует обеспечения определенных возможностей от графической составляющей программной платформы.

В связи с этим принято решение полностью переработать программу, более детально рассмотрев некоторые аспекты разработки базовых структур данных и программных алгоритмов, опираясь на использование новых возможностей современных языков программирования и других инструментов разработки прикладных программ.

2.2 Постановка задачи на проектирование

Разрабатываемая программа предназначена для экспериментальных исследований методов логического вывода делением дизъюнктов, поиска новых классов задач, в которых может применяться логический вывод. Основная цель разрабатываемой программы – поддержка исследований в области логического вывода путем автоматизации процесса решения логических задач с помощью дедуктивного и абдуктивного вывода с построением схемы. Проект может применяться как преподавателями в рамках научной деятельности по данной тематике, так и студентами при изучении дисциплины логического программирования.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 7 |

| | | | | |
|---------------|-----------------|-------------|--------------|----------------|
| Инв. № прокл. | Подпись и дата. | Взам. инв № | Инв. № дубл. | Подпись и дата |
| | | | | |

- вывод информации о выполняемых процедурах в файл или консоль отладки (журнал выполнения).

База знаний, данные выполненных задач и все сопутствующие настройки должны сохраняться в единый бинарный файл с расширением lkb. Необходимо предусмотреть шифрование файла с целью защиты информации базы знаний от

| | | | | |
|---------------|-----------------|-------------|--------------|----------------|
| Инв. № прокл. | Подпись и дата. | Взам. инв № | Инв. № дубл. | Подпись и дата |
| | | | | |

Интерфейс программы должен быть простым и интуитивно понятным. Необходимо предусмотреть использование главного меню программы, выдачу сообщений об успешно завершенных действиях, сообщений об ошибках и предупреждений. В общем случае, предполагается использование следующих элементов интерфейса:

- Цветовая гамма приложения должна быть выполнена в светло-серых тонах, разрешаются темно-зеленый и темно-красный цвета для выделения различных сообщений.

В данном разделе приведено обоснование необходимости разработки программы логического вывода с построением схемы и представлены основные требования к программе. 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 решении тех или иных задач.

3 Математический аппарат и алгоритмы логического вывода

В данном разделе представлено формальное описание методов дедуктивного[3] и абдуктивного логического вывода заключения с построением схемы, в основе которых лежит операция обобщенного деления дизъюнктов. Далее приводятся словесные описания и графические представления программных алгоритмов, основанные на данных методах.

3.1 Дедуктивный логический вывод заключения с построением схемы

Дедуктивный вывод, при проведении которого новые утверждения выступают следствиями из уже имеющихся утверждений, хотя и имеет ограниченную область применения, но надежен при условии истинности посылок. В простейшем случае дедуктивный вывод заключается в установлении факта логического следования заданного заключения из исходных посылок. При успешном завершении вывода может быть получен дополнительный результат в виде схемы вывода, которая наглядно демонстрирует ход рассуждений.

3.1.1 Основные определения

Для описания задачи логического вывода заключений с построением схемы логического вывода введен ряд определений[1].

Определение 1. Задача дедуктивного вывода из множества посылок M заданного заключения d состоит в установлении факта логического следования из посылок M заключения d . При этом, если вывод успешен, то может быть определена логическая цепочка вывода, объясняющая нить рассуждений.

Определение 2. Формула $X_1 \cap X_2 \cap \dots \cap X_K \rightarrow Y$ называется формулой логического следования, символы X_1, X_2, \dots, X_K, Y – литералами, а символ « \rightarrow » – знаком логического следования, который разделяет формулу на левую (антецедент) и правую (консеквент) части. Знаки конъюнкции в формуле могут быть опущены.

Определение 3. Схемой логического вывода называется ориентированный граф, вершинам которого сопоставлены формулы (номера формул) посылок, причем, входящие дуги помечены переменными из левых, а исходящие – из правых частей формул, и вершины соединены между собой в соответствии с суперпозицией их формул в выводе; переменные свободных входящих дуг графа совпадают с переменными из левой, а свободных исходящих дуг – из правой части

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 10 |

формулы заключения (вершина, соответствующая формуле заключения на схеме может не отображаться).

Задачу дедуктивного логического вывода заключений с построением схемы логического вывода можно сформулировать следующим образом. Имеются исходные непротиворечивые посылки, заданные в виде множества дизъюнктов $M = \{D_1, D_2, \dots, D_I\}$. Множество M может включать подмножество однолитеральных дизъюнктов M^F – фактов. Также имеется заключение, представленное множеством выводимых дизъюнктов $m = \{d_1, d_2, \dots, d_T\}$. Таким образом необходимо:

– установить выводимость заключения, представленного множеством дизъюнктов m , из множества исходных посылок, заданных множеством дизъюнктов M ;

– при успешном дедуктивном логическом выводе ($M \Rightarrow m$) сформировать описание O , по которому может быть построена схема логического вывода.

3.1.2 Схема логического вывода

Все исходные секвенции-дизъюнкты предварительно нумеруются, образуя множество номеров секвенций N . Обозначив через A множество различных переменных, используемых в секвенциях, описание схемы можно представить в виде множества литералов: $G' = \{L(j, k); L \in A, j, k \in N\}$, где j – номер секвенции, из вершины которой на схеме выходит дуга, помеченная литералом L , а k – номер секвенции, в вершину которой она входит. По номеру исходной секвенции однозначно определяются литералы, помечающие входящие и выходящие дуги вершины этой секвенции на схеме. В описании схемы используются только положительные (без инверсии) литералы.

Для упрощения формирования описания схемы вывода каждый литерал L дизъюнктов посылок содержит два параметра: $L(k, +)$, где k – номер секвенции, в которую он входит, а «+» – символ вспомогательной переменной, определенной на множестве номеров секвенций N . При этом все отрицательные литералы в дизъюнктах посылок заменяются на положительные в соответствии с равенством: $\neg L(k, +) = L(+, k)$. Каждый литерал L^* дизъюнкта-заключения также содержит два параметра: $L^*(+, j)$, где j – номер секвенции заключения; а все отрицательные литералы заменяются на положительные в соответствии с равенством: $\neg L^*(+, j) = L^*(j, +)$. Кроме того, для упрощения построения схемы логического вывода ее описание представляется в виде семейства множеств $O = \{G_1, G_2, \dots, G_h, \dots, G_H\}$, где G_h – множество литералов, полученных при формировании описания схемы на h -м шаге логического вывода.

| | | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|-----------------|------|------|----------|---------|------|--------------------|----|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата. | | | | | | Лист | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | 11 |

3.1.3 Обобщенное деление дизъюнктов

Множества литералов, входящих в дизъюнкт-посылку D_i и дизъюнкт-заключение d , обозначаются соответственно через $\tilde{D}_i = \{L^i_1, L^i_2, \dots, L^i_j, \dots, L^i_{j_i}\}$ и $\tilde{d} = \{L_1, L_2, \dots, L_k, \dots, L_K\}$, а через \emptyset – пустое множество. Тогда операцию обобщенного деления дизъюнкта D_i на дизъюнкт d , результатом которой является частное a и остаток b , можно записать так: $D_i \% d = \langle a_i, b_i \rangle$.

Частное a_i определяется следующим образом: $a_i = \tilde{D}_i \hat{\cap} \tilde{d}$, где символ « $\hat{\cap}$ » означает специальное пересечение множеств \tilde{D}_i и \tilde{d} . Особенностью специального пересечения является то, что в результирующее множество включаются только те литералы, которые образуются при «склеивании» литералов множеств \tilde{D}_i и \tilde{d} . Под «склеиванием» понимается преобразование пары одноименных литералов $L(+, k) \in \tilde{D}_i$ и $L(j, +) \in \tilde{d}$ (или $L(k, +) \in \tilde{D}_i$ и $L(+, j) \in \tilde{d}$), содержащих в качестве параметра вспомогательную переменную «+», причем в разных позициях, в литерал $L(j, k)$ (или $L(k, j)$), значениями параметров которого являются константы исходных литералов.

При вычислении остатка используется операция специального объединения множеств литералов $\tilde{D} \hat{\cup} a$, особенностью которой является поглощение литералов $L(+, k) \in \tilde{D}$ и $L(j, +) \in \tilde{D}$ литералом $L(j, k) \in \tilde{d} : \{L(+, k)\} \hat{\cup} \{L(j, k)\} = \{L(j, +)\} \hat{\cup} \{L(j, k)\} = \{L(j, k)\}$. Остаток b определяется по следующим правилам:

- если $a = \emptyset$, то $b = 1$;
- если $a \neq \emptyset$, и $(\tilde{D} \hat{\cup} a) - a = \emptyset$, то $b = 0$;
- если $a \neq \emptyset$, и $(\tilde{D} \hat{\cup} a) - a = \tilde{b}$, $\tilde{b} \neq \emptyset$, то $b = L_1 \cup L_2 \cup \dots \cup L_s \cup \dots \cup L_S$, где $L_s \in \tilde{b}$ ($s = 1 \dots S$) и $\tilde{b} = \{L_1, L_2, \dots, L_S\}$.

Например, если $\tilde{D}_1 = \{A(+, 1), B(1, +), E(+, 1)\}$ и $\tilde{d}_1 = \{A(3, +), B(3, +)\}$, то $D_1 \% d_1 = \langle a_1, b_1 \rangle$, где $a_1 = \tilde{D}_1 \hat{\cap} \tilde{d}_1 = \{A(3, 1)\}$ и $b_1 = B(1, +) \cup E(+, 1)$.

3.1.4 Процедура дедуктивного вывода

Метод использует специальную процедуру вывода, названную процедурой обобщенного деления дизъюнктов $w = \langle M, d, q, g, M_1, t \rangle$, в которой:

- $M = \{D_1, D_2, \dots, D_i, \dots, D_l\}$ – множество исходных дизъюнктов;
- $D_i = L^i_1 \cup L^i_2 \cup \dots \cup L^i_j \cup \dots \cup L^i_{j_i}$ – дизъюнкт i -й секвенции, состоящий из литералов $L^i_j \in \{L^i_j(k, +), L^i_j(+, k)\}$;
- $d = L_1 \cup L_2 \cup \dots \cup L_k \cup \dots \cup L_K$ – дизъюнкт заключения, состоящий из литералов $L_k \in \{L_k(+, j), L_k(j, +)\}$;

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 12 |

– q – признак окончания вывода, который может принимать три значения: «0» – вывод завершен успешно, «1» – вывод завершен неудачно, «Р» – требуется продолжение вывода;

– $g = \{L(j, k); L \in A, j, k \in N\}$ – множество частных, которое используется при формировании схемы вывода, при этом j, k – номера секвенций;

– M_1 – новое множество исходных дизъюнктов;

– m – новое множество выводимых дизъюнктов.

Процедура обобщенного деления дизъюнктов применима в случае, если $M \neq \emptyset$ и $J_i, K \geq 1 (i = 1 \dots I)$, в противном случае устанавливается признак $q = 1$. В процедуре выполняются следующие действия:

1) все дизъюнкты исходных секвенций делятся на дизъюнкт выводимой секвенции: $D_i \% d = \langle a_i, b_i \rangle (i = 1 \dots I)$. Образуется начальное множество частных: $g^* = \bigcup_{i=1}^I a_i$. Причем, если хотя бы один остаток $b_i = 0$, то принимается $q = 0$, $g = g^*$, и осуществляется переход к п.7, иначе выполняется следующее действие;

2) если все остатки $b_i = 1$, то вывод невозможен; принимается $q = 1$ и производится переход к п.7, иначе выполняется следующее действие;

3) проверяется наличие однолитеральных дизъюнктов (фактов). Если фактов нет, то принимается $B_k = b_k, k = 1 \dots K$ (где K число остатков, полученных после удаления единичных остатков) и выполняется следующий пункт. Иначе остатки b_k делятся на вспомогательный дизъюнкт r , составленный из литералов фактов: $R_k = b_k \% r = \langle a'_k, b'_k \rangle, k = 1 \dots K$. При этом формируются упрощенные остатки: $B_k = b_k$, если $b'_k = 1$ и $B_k = b'_k$ – в противном случае. Корректируется множество частных: $g = g^* \cup \bigcup_{k=1}^K a'_k$. Причем, если хотя бы один остаток $b'_k = 0$, то принимается $q = 0$ и осуществляется переход к п.7, иначе выполняется следующий пункт;

4) составляется выражение $F = \bigcap_{k=1}^K B_k$, которое упрощается путем перемножения дизъюнктов и исключения конъюнкций, содержащих контрарные пары литералов вида $L(u, +) \cap L(+, v) (L \cap \neg L)$ и сомножители $0 \cap L(+, w), 0 \cap L(w, +)$. При этом учитывается, что $L(u, +) \cap L(+, v) = 0$. Если преобразуемое таким образом выражение оказывается равным нулю ($F = 0$), то вывод успешно завершается. В этом случае принимается $q = 0$, производится операция «склеивания» каждой контрарной пары литералов $L(u, +)$ и $L(+, v)$ в литерал $L(u, v)$, который добавляется во множество частных g , и осуществляется переход к п.7, в противном случае выполняется следующий пункт;

5) формула F представляет собой дизъюнкцию конъюнкций литералов (дизъюнктивную форму). Эта формула преобразуется в конъюнктивную форму путем замены операций на соответствующие им дуальные: конъюнкции на дизъюнкцию, а дизъюнкции на конъюнкцию. В результате этого получается

| | | | | | | | | | | | | | | | | | | | | | |
|--|----------------|----------|--------------|------|--------------------|------|----------------|--|------|------|----------|---------|------|--------------------|------|--|--|--|--|--|----|
| Инв. № прокл. | Подпись и дата | | Инв. № докл. | | Взам. инв. № | | Подпись и дата | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |
| <table border="1"> <tr> <td>Изм.</td> <td>Лист</td> <td>№ докум.</td> <td>Подпись</td> <td>Дата</td> <td rowspan="2">ТПЖА.230101.016 ПЗ</td> <td>Лист</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td>13</td> </tr> </table> | | | | | | | | | Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | Лист | | | | | | 13 |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | Лист | | | | | | | | | | | | | | | |
| | | | | | | 13 | | | | | | | | | | | | | | | |

формула $F^* = \bigcap_{t=1}^T d_t$, где d_t – дизъюнкт, совокупность которых в формуле F^* рассматривается как множество новых выводимых дизъюнктов и включается во множество $m = \{d_t, t = 1 \dots T\}$. Значение признака окончания вывода устанавливается $q = P$;

б) определяется новое множество исходных дизъюнктов: $M_1 = M - M_0$, где M_0 – подмножество множества M , в которое входят дизъюнкты, имеющие отличные от единицы остатки b_i ($i = 1 \dots I$) при делении на дизъюнкт заключения. Причем, если $M_1 = \emptyset$, то принимается $q = 1$;

7) для процедуры $w = \langle M, d, q, g, M_1, m \rangle$ фиксируются результаты.

Дедуктивный логический вывод заключений с построением схемы представляется многократным применением w -процедур и состоит из ряда шагов. На каждом шаге вывода w -процедуры используются новые исходные дизъюнкты-посылки и дизъюнкты-заключения, образованные на предыдущем шаге. Процесс вывода заканчивается, когда на очередном шаге обнаруживается дизъюнкт, для которого вывод невозможен ($q = 1$), либо для всех выводимых на данном шаге дизъюнктов, будут сформированы признаки успешного завершения вывода ($q = 0$). При этом подмножество фактов M_F включается во множество исходных дизъюнктов M только на первом шаге метода. На всех последующих шагах метода факты не входят во множества исходных дизъюнктов. Описание логической схемы представляется в виде семейства O общих множеств частных G , которые формируются на каждом шаге за счет объединения множеств частных g , получаемых в результате деления дизъюнктов в w -процедурах.

3.2 Абдуктивный логический вывод заключения с построением схемы

Задачу абдуктивного логического вывода заключения с построением схемы логического вывода можно сформулировать следующим образом. Имеются исходные непротиворечивые посылки, заданные в виде множества дизъюнктов $M = \{D_1, D_2, \dots, D_I\}$. Множество M может включать подмножество однолитеральных дизъюнктов M^F – фактов. Также имеется заключение, представленное множеством выводимых дизъюнктов $m = \{d_1, d_2, \dots, d_T\}$. Таким образом необходимо:

– установить выводимость заключения, представленного множеством дизъюнктов m , из множества исходных посылок, заданных множеством дизъюнктов M ;

– в случае невыводимости заключения сформировать множество дополнительных посылок, которое вместе с множеством M обеспечивают успешный логический вывод заключения m ;

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 14 |

– если множество дополнительных посылок пусто, то абдуктивный вывод завершается неудачей, в противном случае необходимо провести анализ полученных дополнительных посылок и установить их непротиворечивость со множеством исходных посылок; при установлении непротиворечивости дополнительных посылок абдуктивный логический вывод завершается успешно, иначе – неудачно;

– при успешном абдуктивном логическом выводе ($M \Rightarrow m$) сформировать описание O , по которому может быть построена схема логического вывода.

Для описания метода абдуктивного вывода с построением схемы используются положения, принятые в разделах 3.1.1, 3.1.2 и 3.1.3.

3.2.1 Операция группового деления дизъюнктов

Операция группового деления дизъюнкта d на множество дизъюнктов $M_0 = \{D_g, g = 1 \dots G\}$, результатом которой является остаток ∂ , записывается следующим образом: $d \div M_0 = \partial$.

Через $\tilde{D}_g = \{L^g_j, j = 1 \dots J_g\}$, $\tilde{d} = \{L_k, k = 1 \dots K\}$ обозначаются множества литералов, входящих соответственно в дизъюнкты D_g и d , а через \emptyset - пустое множество. При этом символ « \div » используется как знак разности множеств, при котором литералы $L(+, k) \in \tilde{d}$ и $L(j, +) \in \tilde{D}_g$ (или $L(k, +) \in \tilde{d}$ и $L(+, j) \in \tilde{D}_g$) отбрасываются и оставшиеся литералы в \tilde{d} образуют остаток ∂ . Таким образом остаток ∂ определяется:

- если $\tilde{d} \cap (\cup_{g=1}^G \tilde{D}_g) = \emptyset$, то $\partial = 1$;
- если $\tilde{d} \cap (\cup_{g=1}^G \tilde{D}_g) \neq \emptyset$ и $\tilde{d} - (\cup_{g=1}^G \tilde{D}_g) = \emptyset$, то $\partial = 0$;
- если $\tilde{d} \cap (\cup_{g=1}^G \tilde{D}_g) \neq \emptyset$ и $\tilde{d} - (\cup_{g=1}^G \tilde{D}_g) = \tilde{\partial}$, $\tilde{\partial} \neq \emptyset$, то $\partial = L_1 \cup L_2 \cup \dots \cup L_S \cup \dots \cup L_S$, где $L_s \in \tilde{\partial}$ ($s = 1 \dots S$) и $\tilde{\partial} = \{L_1, L_2, \dots, L_S\}$.

3.2.2 Процедура абдуктивного вывода

Основной составляющей абдуктивного логического вывода с построением схемы является специальная процедура $\dot{w} = \langle M, d, q, p, g, M_1, m, \partial \rangle$, в основе которой лежит операция обобщенного деления дизъюнктов:

- $M = \{D_1, D_2, \dots, D_i, \dots, D_I\}$ – множество исходных дизъюнктов;
- $D_i = L^i_1 \cup L^i_2 \cup \dots \cup L^i_j \cup \dots \cup L^i_{j_i}$ – дизъюнкт i -й секвенции, состоящий из литералов $L^i_j \in \{L^i_j(k, +), L^i_j(+, k)\}$;

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инф. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 15 |

- $d = L_1 \cup L_2 \cup \dots \cup L_k \cup \dots \cup L_K$ – дизъюнкт заключения, состоящий из литералов $L_k \in \{L_k(+, j), L_k(j, +)\}$;
- q – признак наличия решения, который может принимать два значения: «0» – решение есть, «1» – решений нет;
- p – признак возможности продолжения вывода: «0» – продолжение вывода возможно, «1» – продолжение вывода невозможно;
- $g = \{L(j, k); L \in A, j, k \in N\}$ – множество частных, которое используется при формировании схемы вывода, при этом j, k – номера секвенций;
- M_1 – новое множество исходных дизъюнктов;
- m – новое множество выводимых дизъюнктов;
- ∂ – дополнение, представляющее собой вспомогательный остаток, который затем используется при формировании дизъюнктов дополнительных посылок.

Процедура $\ddot{w} = \langle M, d, q, p, g, M_1, m, \partial \rangle$ отличается от процедуры, используемой для дедуктивного вывода, $w = \langle M, d, q, g, M_1, m \rangle$, тем, что в ней применяется пара признаков q и p , а также вычисляется дополнение ∂ .

Процедура абдуктивного логического вывода делением дизъюнктов применима в случае, если $M \neq \emptyset$ и $J_i, K \geq 1 (i = 1 \dots I)$, в противном случае устанавливается признак $q = 1$. В процедуре выполняются следующие действия:

1) все дизъюнкты исходных секвенций делятся на дизъюнкт выводимой секвенции: $D_i \% d = \langle a_i, b_i \rangle (i = 1 \dots I)$. Образуется начальное множество частных: $g^* = \bigcup_{i=1}^I a_i$. Причем, если хотя бы один остаток $b_i = 0$, то принимается $q = 0, p = 1, g = g^*, \partial = 0$, и осуществляется переход к п.8, иначе выполняется следующее действие;

2) если все остатки $b_i = 1$, то вывод невозможен; принимается $q = p = 1, \partial = d$ и производится переход к п.8, иначе выполняется следующее действие;

3) проверяется наличие однолитеральных дизъюнктов (фактов). Если фактов нет, то принимается $B_k = b_k, k = 1 \dots K$ (где K число остатков, полученных после удаления единичных остатков) и выполняется следующий пункт. Иначе остатки b_k делятся на вспомогательный дизъюнкт r , составленный из литералов фактов: $R_k = b_k \% r = \langle a'_k, b'_k \rangle, k = 1 \dots K$. При этом формируются упрощенные остатки: $B_k = b_k$, если $b'_k = 1$ и $B_k = b'_k$ – в противном случае. Корректируется множество частных: $g = g^* \cup \bigcup_{k=1}^K a'_k$. Причем, если хотя бы один остаток $b'_k = 0$, то принимается $q = 0, p = 1, \partial = 0$ и осуществляется переход к п.8, иначе выполняется следующий пункт;

4) составляется выражение $F = \bigcap_{k=1}^K B_k$, которое упрощается путем перемножения дизъюнктов и исключения конъюнкций, содержащих контрарные пары литералов вида $L(u, +) \cap L(+, v) (L \cap \neg L)$ и сомножители $0 \cap L(+, w), 0 \cap L(w, +)$. При этом учитывается, что $L(u, +) \cap L(+, v) = 0$. Если

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инф. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 16 |

преобразуемое таким образом выражение оказывается равным нулю ($F = 0$), то вывод успешно завершается. В этом случае принимается $q = 0$, $p = 1$, $\partial = 0$ и производится операция «склеивания» каждой контрарной пары литералов $L(u, +)$ и $L(+, v)$ в литерал $L(u, v)$, который добавляется во множество частных g , и осуществляется переход к п.8, в противном случае выполняется следующий пункт;

5) формула F представляет собой дизъюнкцию конъюнкций литералов (дизъюнктивную форму). Эта формула преобразуется в конъюнктивную форму путем замены операций на соответствующие им дуальные: конъюнкции на дизъюнкцию, а дизъюнкции на конъюнкцию. В результате этого получается формула $F^* = \bigcap_{t=1}^T d_t$, где d_t – дизъюнкт, совокупность которых в формуле F^* рассматривается как множество новых выводимых дизъюнктов и включается во множество $m = \{d_t, t = 1 \dots T\}$. Устанавливаются значения признаков $q = 1$ и $p = 0$;

6) определяется новое множество исходных дизъюнктов: $M_1 = M - M_0$, где M_0 – подмножество множества M , в которое входят дизъюнкты, имеющие отличные от единицы остатки b_i ($i = 1 \dots I$) при делении на дизъюнкт заключения. Причем, если $M_1 = \emptyset$, то принимается $q = 1$, $p = 1$;

7) вычисляется дополнение ∂ для дизъюнкта d с помощью операции группового деления дизъюнктов: $\partial = d \div M_0$, при этом литералы множества $\tilde{d} = \{L_1, L_2, \dots, L_S\}$, образующие дизъюнкт ∂ , сохраняют за собой значения параметров ($L(+, k)$ или $L(k, +)$), которые они имеют при вхождении в k -й дизъюнкт множества M_0 .

8) для процедуры $\dot{w} = \langle M, d, q, p, g, M_1, m, \partial \rangle$ фиксируются результаты.

В процессе вывода формируется формула дополнительных посылок, которая принимается к рассмотрению в случае неуспешного завершения вывода.

3.2.3 Формирование дополнительных посылок

Абдуктивный логический вывод заключений с построением схемы представляется многократным применением \dot{w} -процедур и состоит из ряда шагов. На каждом шаге вывода \dot{w} -процедуры используются новые исходные дизъюнкты-посылки и дизъюнкты-заключения, образованные на предыдущем шаге, строится формула дополнительных посылок и формируется значение признака успешного завершения абдуктивного логического вывода. Процесс вывода заканчивается, когда для каждого на очередном шаге для каждого выводимого дизъюнкта будет получено решение ($q = 0$) либо будет установлено что продолжение вывода невозможно ($p = 1$).

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 17 |

Для описания процесса формирования формулы дополнительных посылок Ψ вводятся обозначения: h – номер шага вывода, T – число выполняемых \ddot{w} -процедур на h -м шаге вывода. Формула дополнительных посылок имеет вид $\Psi_h = \bigcap_{i=1}^T ((\partial_i \cup p_i \Psi_{h-1}) \cap q_i)$. По окончанию вывода данная конъюнкция представляет собой множество дополнительных дизъюнктов $M^D = \{D_{I+n}, n = 1 \dots N\}$, где I – число исходных дизъюнктов посылок для задачи, N – число дополнительных дизъюнктов-посылок. Причем, если процесс абдуктивного вывода завершается успешно, то дополнительные посылки не требуются.

В случае неуспешного завершения абдуктивного вывода, после формирования дополнительных дизъюнктов, осуществляется проверка на их противоречивость с множеством исходных дизъюнктов. Для этого составляется конъюнктивное выражение вида: $f = D_{I+1} \cap D_{I+2} \cap \dots \cap D_{I+N}$, которое упрощается путем перемножения дизъюнктов и исключения конъюнкций, содержащих контрарные пары литералов вида $L(u, +) \cap L(+, v)$ ($L \cap \neg L$) и сомножители $0 \cap L(+, w)$, $0 \cap L(w, +)$. При этом учитывается, что $L(u, +) \cap L(+, v) = 0$. Если $f = 0$, то множество дополнительных дизъюнктов противоречиво. Принимается $M^D = \emptyset$ и абдуктивный вывод завершается неудачно. Иначе формула f инвертируется и полученные конъюнктивные сомножители образуют множество m' выводимых дизъюнктов для проверочного дедуктивного вывода, для которого множеством исходных посылок является множество M . Успешное завершение проверочного дедуктивного вывода показывает противоречивость объединенного множества дизъюнктов посылок. Принимается $M^D = \emptyset$ и абдуктивный вывод завершается неудачно. В противном случае, если проверочный дедуктивный вывод завершается неудачно, то объединенное множество дизъюнктов посылок непротиворечиво, при этом множество M^D содержит дизъюнкты дополнительных посылок и абдуктивный вывод завершается успешно.

Описание логической схемы представляется в виде семейства O общих множеств частных G , которые формируются на каждом шаге за счет объединения множеств частных g , получаемых в результате деления дизъюнктов в \ddot{w} -процедурах. При этом для множества дизъюнктов дополнительных посылок M^D необходимо выполнить следующие действия:

- каждому n дизъюнкту множества M^D присвоить номер $o = I + n$ ($n = 1 \dots N$);
- для каждого литерала L дизъюнкта D_o из множества M^D выполнить замену параметра со значением «+» на параметр со значением o : $L(+, k) \rightarrow L(o, k)$, $L(k, +) \rightarrow L(k, o)$;
- сформировать множество литералов $O^D = \{\tilde{D}_1, \tilde{D}_2, \dots, \tilde{D}_n\}$, где \tilde{D}_n – множество литералов, входящих в дизъюнкт n множества M^D .

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|-----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата. | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 18 |

После этого множество O^D может быть объединено с семейством O и использоваться для построения дополнительных элементов схемы логического вывода.

Применение описанного метода можно рассмотреть на следующем примере. «Если я открою бензобак машины и залью бензин, то машина будет заправлена. Если я почию машину, то машина будет исправна. Если машина сможет двигаться, то я смогу попасть на работу. Следовательно, если я открою бензобак машины, залью бензин и почию машину, то я смогу попасть на работу».

Описанные правила можно записать в символическом виде: 1) $A \cap B \rightarrow C$; 2) $D \rightarrow E$; 3) $F \rightarrow G$. При этом заключение: -1) $A \cap B \cap D \rightarrow G$ (номер -1 вводится специально для заключения, чтобы разграничить индексацию исходных посылок и заключения). Правила и заключение преобразуются в дизъюнкты и выполняется абдуктивный логический вывод, результаты которого представлены на рисунке 3.1.

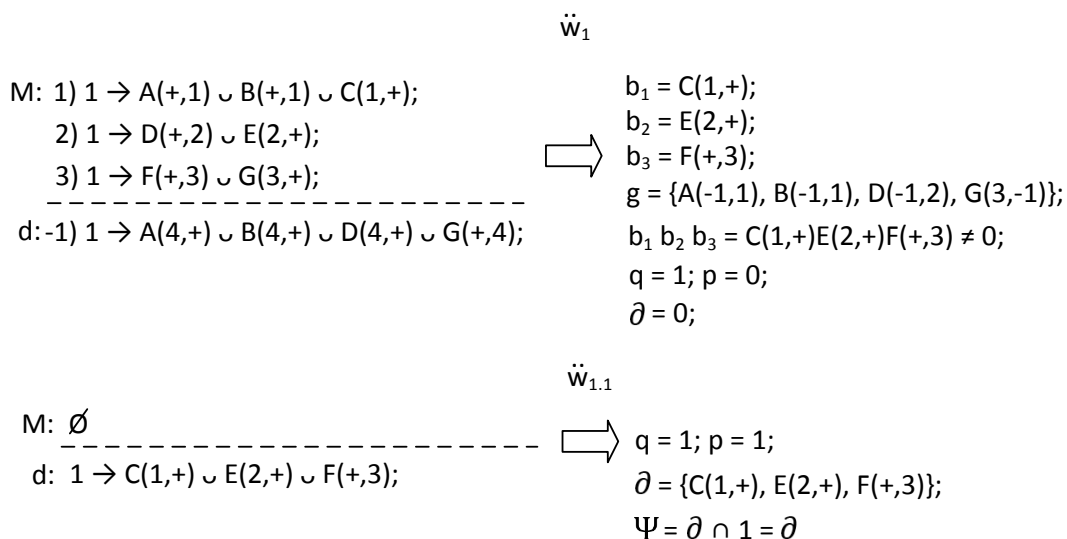


Рисунок 3.1 – Результаты абдуктивного логического вывода

Как видно из рисунка 3.1, абдуктивный логический вывод завершается неудачей на втором шаге. При этом сформирована формула дополнительных посылок $\Psi = C(1,+) \cup E(2,+) \cup F(+,3)$, которая содержит один дизъюнкт. Данный дизъюнкт не может быть логически выведен из исходного множества посылок, и, следовательно, его добавление не приведет к противоречивому множеству исходных посылок. Таким образом абдуктивный логический вывод успешно завершается. После этого для найденного дополнительного дизъюнкта выполняются следующие действия:

- дизъюнкту присваивается номер 4, поскольку число исходных посылок равно трем;
- модифицируются параметры литералов дизъюнкта, значения «+» заменяются на присвоенный номер дизъюнкта: $C(1,4) \cup E(2,4) \cup F(4,3)$;

| | | | | | | |
|---------------|----------------|--------------|--------------|----------------|--|------|
| Инф. № прокл. | Подпись и дата | Взам. инб. № | Инф. № дубл. | Подпись и дата | <div style="font-size: 24px; font-weight: bold; margin-bottom: 10px;">ТПЖА.230101.016 ПЗ</div> <div style="display: flex; justify-content: space-between; font-size: 12px;"> Изм. Лист № докум. Подпись Дата </div> | Лист |
| | | | | | | |
| | | | | | | |
| | | | | | | 19 |

- литералы дизъюнкта образуют дополнительное множество O^D , которое может быть включено в множество O для построения схемы вывода.

Схема, построенная с помощью объединения множеств O и O^D для абдуктивного логического вывода представлена на рисунке 3.2.

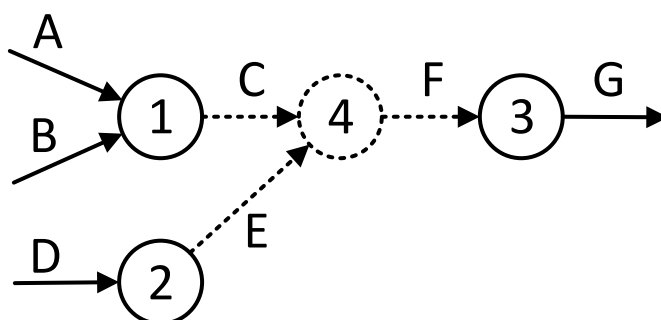


Рисунок 3.2 – Схема абдуктивного логического вывода

Таким образом, можно сформулировать дополнительное правило для исходной задачи: «Если машина будет заправлена и исправна, то машина сможет двигаться.»

3.3 Разработка программных алгоритмов логического вывода делением ДИЗЬЮНКТОВ

Проанализировав описание математического аппарата для задачи логического вывода можно выделить следующие программные алгоритмы: алгоритм обобщенного деления дизъюнктов, алгоритм выполнения процедуры дедуктивного логического вывода, алгоритм выполнения процедуры абдуктивного вывода, алгоритм построения яруса дерева дедуктивного логического вывода, алгоритм построения яруса дерева абдуктивного логического вывода, алгоритм формирования дерева логического вывода, алгоритм формирования схемы логического вывода.

Операция обобщенного деления дизъюнктов формулируется следующим образом:

- 1) создать дизъюнкт-остаток;
- 2) создать дизъюнкт-частное;

3) последовательно сопоставить дизъюнкты делимого и делителя: если обнаружена контрарная пара литералов, то выполняется операция «склеивания» данной пары и соответствующий литерал добавляется в дизъюнкт-частное, иначе, если литерал дизъюнкта-делимого не имеет контрарной пары в дизъюнкте-делимом, то он добавляется в дизъюнкт-остаток;

- 4) конец процедуры.

Графическое представление алгоритма приведено на рисунке 3.3.

| | | | | |
|---------------|----------------|--------------|--------------|----------------|
| Инв. № прокл. | Подпись и дата | Взам. инв. № | Инв. № дубл. | Подпись и дата |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

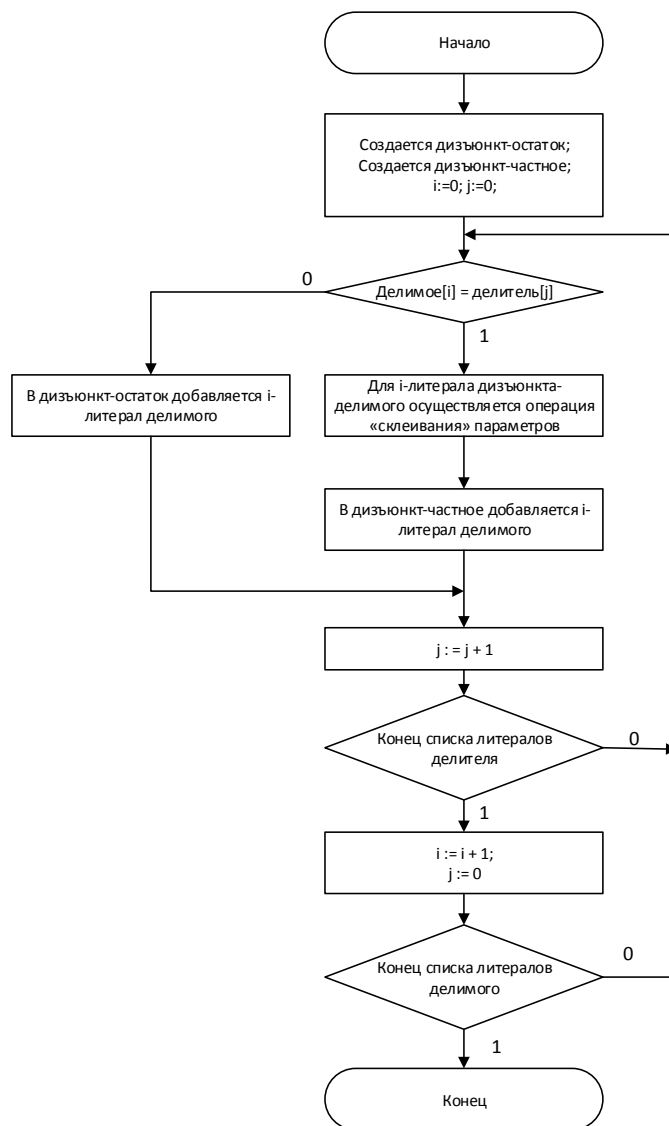


Рисунок 3.3 – Графическое представление программного алгоритма обобщенного деления дизъюнктов

Алгоритм выполнения процедуры дедуктивного логического вывода заключается в многократном применении операции деления дизъюнктов и последующем анализе остатков:

- 1) для каждого из множества исходных дизъюнктов выполнить операцию деления дизъюнктов на дизъюнкт-следствие;
- 2) последовательно проанализировать полученное после деления множество остатков: если количество литералов в дизъюнкте-остатке равно количеству литералов в дизъюнкте-делимом, то остаток исключается из множества остатков;
- 3) если множество остатков пусто, то установить флаг неудачного завершения процедуры и переход к п.13;
- 4) если один из остатков равен нулю, то установить флаг успешного завершения процедуры и переход к п.13;
- 5) если текущая процедура логического вывода выполняется на первом шаге вывода, то переход к п.6, иначе переход к п.9;

6) сформировать дизъюнкт, составленный фактов следующим образом: последовательно проанализировать каждый дизъюнкт из множества исходных, в случае если дизъюнкт состоит только из одного литерала – добавить данный литерал в дизъюнкт фактов;

7) осуществить деление множества дизъюнктов-остатков на дизъюнкт, составленный из литералов фактов, при этом новое множество остатков и множество частных формируется аналогично п.1 и п.2;

8) если один из остатков равен нулю, то установить флаг успешного завершения процедуры и переход к п.13;

9) сформировать и упростить выражение F (конъюнкцию дизъюнктов-остатков), путем перемножения дизъюнктов и исключения контрарных литералов;

10) если конъюнкция остатков равна нулю, то установить флаг успешного завершения процедуры и переход к п.13, иначе установить флаг необходимости продолжения вывода и переход к п.11;

11) преобразовать выражение F в конъюнктивную форму путем замены конъюнкций на дизъюнкции, а дизъюнкций на конъюнкции;

12) выполнить коррекцию множества исходных дизъюнктов: исключаются все дизъюнкты, по которым после деления были сформированы непустые дизъюнкты-остатки;

13) конец процедуры.

Графическое представление программного алгоритма процедуры логического вывода приведено на рисунке 3.4.

Алгоритм выполнения процедуры абдуктивного логического вывода заключается в многократном применении операции деления дизъюнктов, анализе остатков от деления и вычислении дополнения:

1) для каждого из множества исходных дизъюнктов выполнить операцию деления дизъюнктов на дизъюнкт-следствие;

2) последовательно проанализировать полученное после деления множество остатков: если количество литералов в дизъюнкте-остатке равно количеству литералов в дизъюнкте-делимом, то остаток исключается из множества остатков;

3) если множество остатков пусто, то установить флаг неудачного завершения процедуры и флаг невозможности продолжения вывода, и осуществить переход к п.13;

4) если один из остатков равен нулю, то установить флаг успешного завершения процедуры и флаг невозможности продолжения вывода, и осуществить переход к п.14;

5) если текущая процедура логического вывода выполняется на первом шаге вывода, то переход к п.6, иначе переход к п.9;

6) сформировать дизъюнкт, составленный фактов следующим образом: последовательно проанализировать каждый дизъюнкт из множества исходных, в случае если дизъюнкт состоит только из одного литерала – добавить данный литерал в дизъюнкт фактов;

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 22 |

7) осуществить деление множества дизъюнктов-остатков на дизъюнкт, составленный из литералов фактов, при этом новое множество остатков и множество частных формируется аналогично п.1 и п.2;

8) если один из остатков равен нулю, то установить флаг успешного завершения процедуры, флаг невозможности продолжения вывода, и осуществить переход к п.14;

9) сформировать и упростить выражение F (конъюнкцию дизъюнктов-остатков), путем перемножения дизъюнктов и исключения контрарных литералов;

10) если конъюнкция остатков равна нулю, то установить флаг успешного завершения процедуры и флаг невозможности продолжения вывода, и осуществить переход к п.14, иначе установить флаг необходимости продолжения вывода, флаг неуспешного завершения процедуры, и осуществить переход к п.11;

11) преобразовать выражение F в конъюнктивную форму путем замены конъюнкций на дизъюнкции, а дизъюнкций на конъюнкции;

12) выполнить коррекцию множества исходных дизъюнктов: исключаются все дизъюнкты, по которым после деления были сформированы непустые дизъюнкты-остатки;

13) с помощью операции группового деления дизъюнктов разделить выводимый дизъюнкт на дизъюнкт, составленный из литералов исходных дизъюнктов, по которым после выполнения п.1 были сформированы непустые остатки; полученный при этом остаток принять как дополнение в рамках выполняемой процедуры;

14) конец процедуры.

Графическое представление программного алгоритма не приводится, поскольку имеет лишь небольшие отличия от аналогичного алгоритма для дедуктивного вывода.

Алгоритм построения яруса дерева дедуктивного логического вывода подразумевает параллельное выполнение процедур одного шага вывода с последующим анализом результатов всех выполненных процедур:

1) создать процедуры логического вывода, при этом количество создаваемых процедур равно количеству дизъюнктов-следствий для текущего шага вывода;

2) выполнить анализ результатов всех выполненных процедур на текущем шаге: если одна из процедур завершена неудачно, то установить флаг неудачно заверщенного шага вывода; если одна из процедур завершена с флагом необходимости продолжения вывода, то установить флаг необходимости продолжения вывода для текущего шага, иначе установить флаг успешно заверщенного шага вывода;

3) конец процедуры.

Графическое представление программного алгоритма построения яруса дерева дедуктивного логического вывода приведено на рисунке 3.5.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прадл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 23 |

Алгоритм построения яруса дерева абдуктивного логического вывода подразумевает параллельное выполнение процедур одного шага вывода с последующим анализом результатов всех выполненных процедур:

1) создать процедуры логического вывода, при этом количество создаваемых процедур равно количеству дизъюнктов-следствий для текущего шага вывода;

2) выполнить анализ результатов всех выполненных процедур на текущем шаге: если одна из процедур завершена с флагом необходимости продолжения вывода, то установить флаг необходимости продолжения вывода для текущего шага, иначе установить флаг невозможности продолжения вывода;

3) конец процедуры.

Графическое представление программного алгоритма не приводится, поскольку имеет лишь небольшие отличия от аналогичного алгоритма для дедуктивного вывода.

Алгоритм построения дерева логического вывода состоит в многократном формировании ярусов дерева, до тех пор, пока последний ярус не будет завершен с флагом невозможности продолжения вывода:

1) сформировать процедуру первого шага и запустить на выполнение;

2) если требуется продолжение вывода, то переход к п.3, иначе переход к п.4

3) сформировать и выполнить процедуры для следующего яруса дерева из порожденных процедур предыдущего, осуществить переход к п.2;

4) флаг успешного или неудачного вывода установить равным аналогичному флагу последнего завершенного шага;

5) конец построения дерева.

Графическое представление программного алгоритма построения дерева логического вывода приведено на рисунке 3.6.

На основе успешно завершенного логического вывода строится схема вывода путем анализа множества частных каждой из процедур вывода:

1) если вершина с идентификатором, равным левому параметру литерала частного уже присутствует в схеме, то переход к п.3, иначе переход к п.2;

2) создать вершину с идентификатором, равным левому параметру литерала частного и добавить ее в схему;

3) если вершина с идентификатором, равным правому параметру литерала частного уже присутствует в схеме, то осуществляется переход к п.5, иначе переход к п.4;

4) создать вершину с идентификатором, равным правому параметру литерала частного и добавить ее в схему;

5) если дуга, помеченная литералом, полностью идентичным литералу частного, уже присутствует в схеме, то переход к п.7, иначе переход к п.6;

6) создать новую дугу, помеченную литералом частного, начальной и конечной вершиной которого являются присутствующие в схеме вершины с идентификаторами, равными левому и правому параметрам литерала частного соответственно;

7) конец построения схемы.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 24 |

Графическое представление программного алгоритма построения схемы логического вывода приведено на рисунке 3.7.

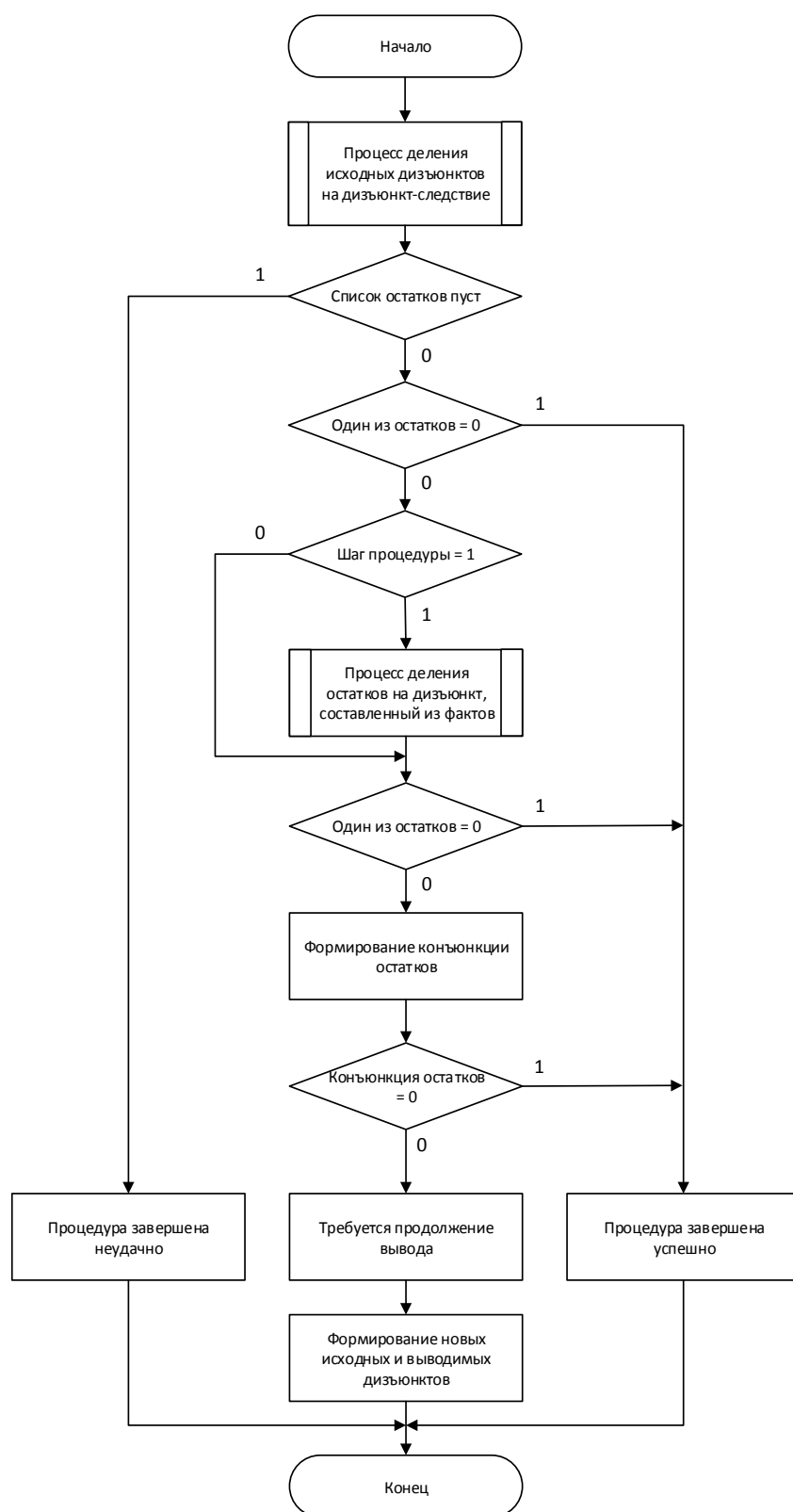


Рисунок 3.4 – Графическое представление программного алгоритма процедуры деления дизъюнктов

| | |
|----------------|----------------|
| Инф. № прокл. | Подпись и дата |
| Взам. инф. № | Инф. № дубл. |
| Подпись и дата | Подпись и дата |

| | | | | |
|------|------|----------|---------|------|
| Изм. | Лист | № докум. | Подпись | Дата |
|------|------|----------|---------|------|

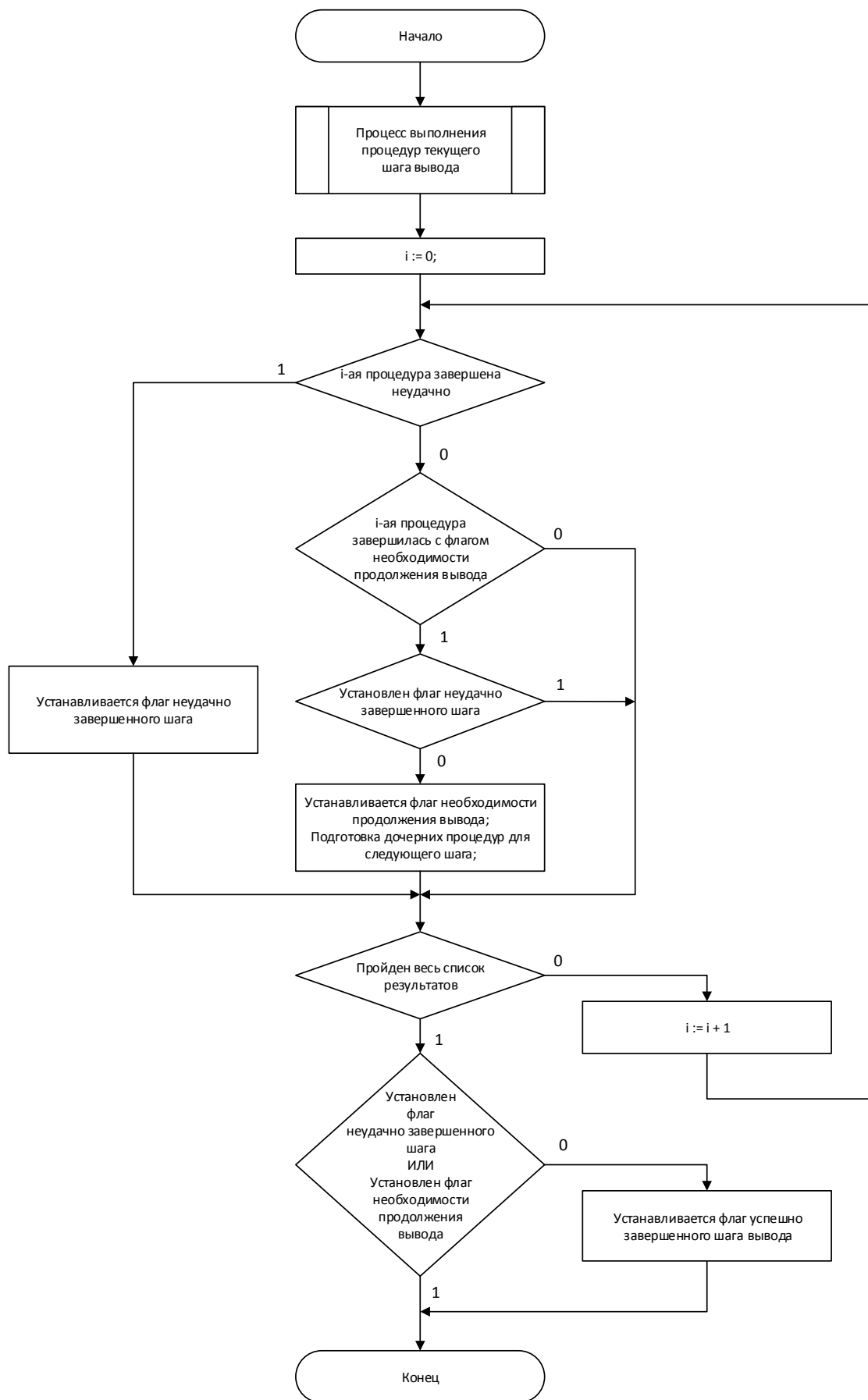


Рисунок 3.5 – Графическое представление программного алгоритма построения яруса дерева логического вывода

| | | | | |
|---------------|----------------|--------------|--------------|----------------|
| Инв. № прокл. | Подпись и дата | Взам. инв. № | Инв. № дубл. | Подпись и дата |
| | | | | |

| | | | | |
|------|------|----------|---------|------|
| Изм. | Лист | № докум. | Подпись | Дата |
| | | | | |

ТПЖА.230101.016 ПЗ

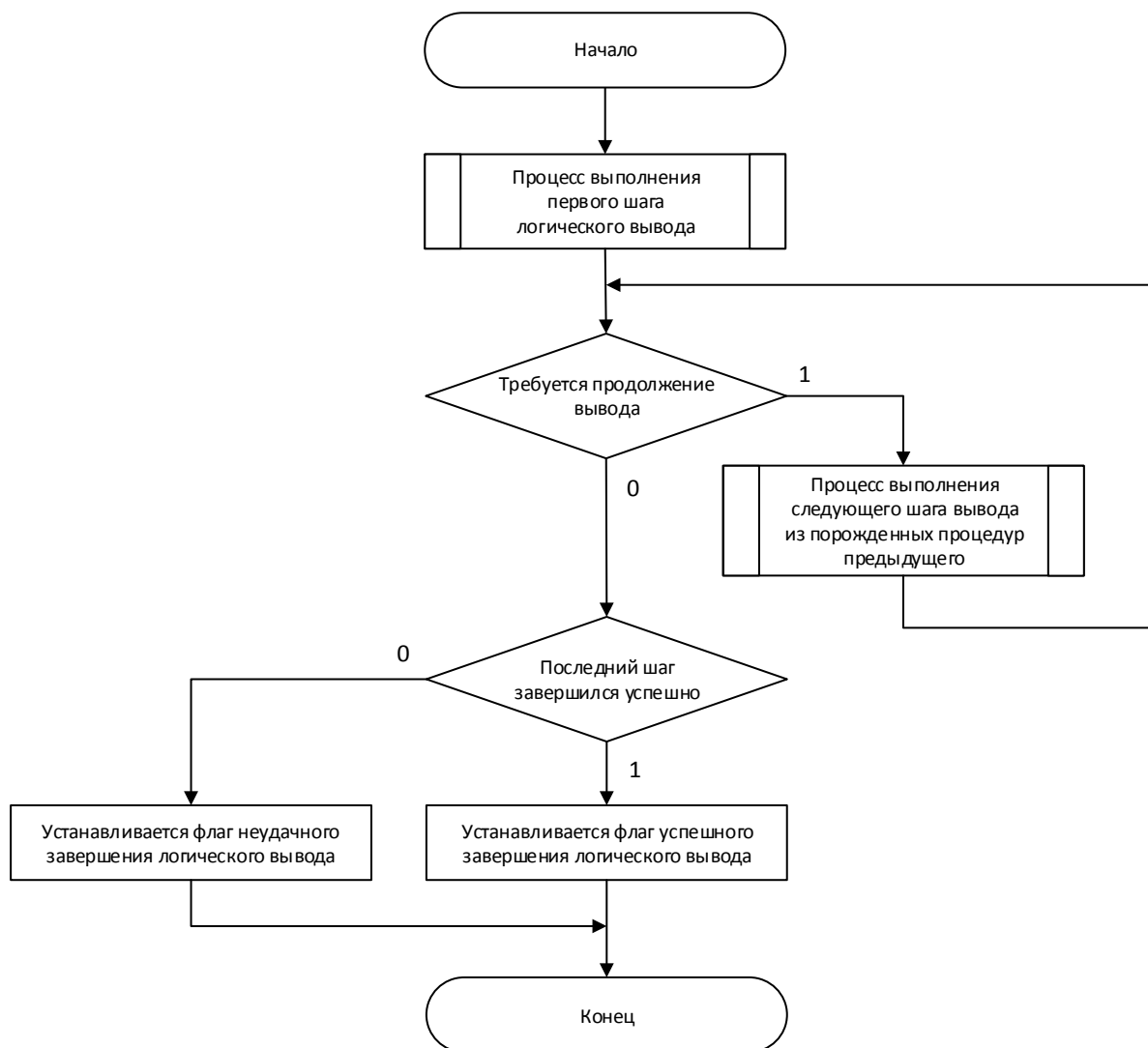


Рисунок 3.6 – Графическое представление программного алгоритма построения дерева логического вывода

| | |
|----------------|----------------|
| Инф. № прораб. | Подпись и дата |
| Взам. инж. № | Инж. № дубл. |
| Подпись и дата | Инж. № дубл. |
| Инж. № прораб. | Подпись и дата |

| | | | | |
|------|------|----------|---------|------|
| Изм. | Лист | № докум. | Подпись | Дата |
|------|------|----------|---------|------|

ТПЖА.230101.016 ПЗ

| | | | | |
|---------------|-----------------|--------------|--------------|----------------|
| Инф. № прокл. | Подпись и дата. | Взам. инф. № | Инф. № дубл. | Подпись и дата |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

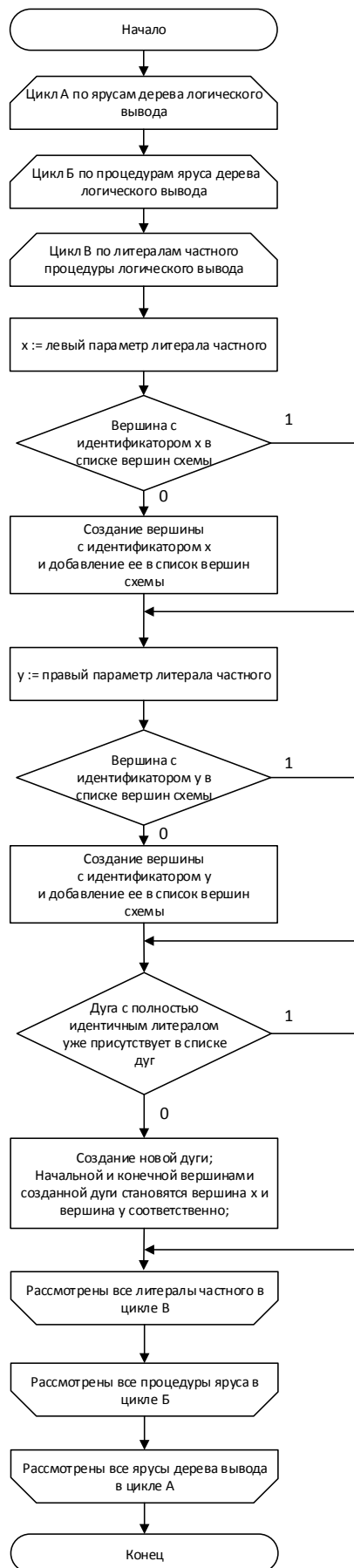


Рисунок 3.7 – Графическое представление программного алгоритма построения схемы логического вывода

Выводы

Рассмотрен метод дедуктивного логического вывода заключения с построением схемы, который позволяет не только установить выводимость заключения из множества исходных посылок, но и в случае успешного вывода сформировать описание схемы логического вывода, по которому она может быть построена. Схема логического вывода позволяет проанализировать процесс развития вывода, определить возможные нити вывода с выделением необходимых фактов и правил[4].

Разработан метод абдуктивного логического вывода заключения с построением схемы, отличающийся от известных методов применением процедуры обобщенного деления дизъюнктов, а также тем, что с помощью специальной процедуры полученные в результате абдуктивного вывода дополнительные посылки преобразуются в дополнительные элементы схемы логического вывода.

Приведены программные алгоритмы, которые являются основой для проектирования структуры библиотеки логического вывода и сопутствующих программных модулей, а также для написания программного кода. При разработке программы предпочтение отдается объектно-ориентированному подходу.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 29 |

4 Разработка программы дедуктивного и абдуктивного логического вывода

В данном разделе рассмотрен подход к разработке программного продукта, представлено описание структуры, приведено детальное описание алгоритмов работы каждого из функциональных блоков программы с примерами исходного кода на языке Java.

4.1 Обобщенная структура программы

Разрабатываемая программа, обобщенная структура которой представлена на рисунке 4.1, состоит из нескольких взаимосвязанных модулей, каждый из которых обеспечивает определенный функционал:

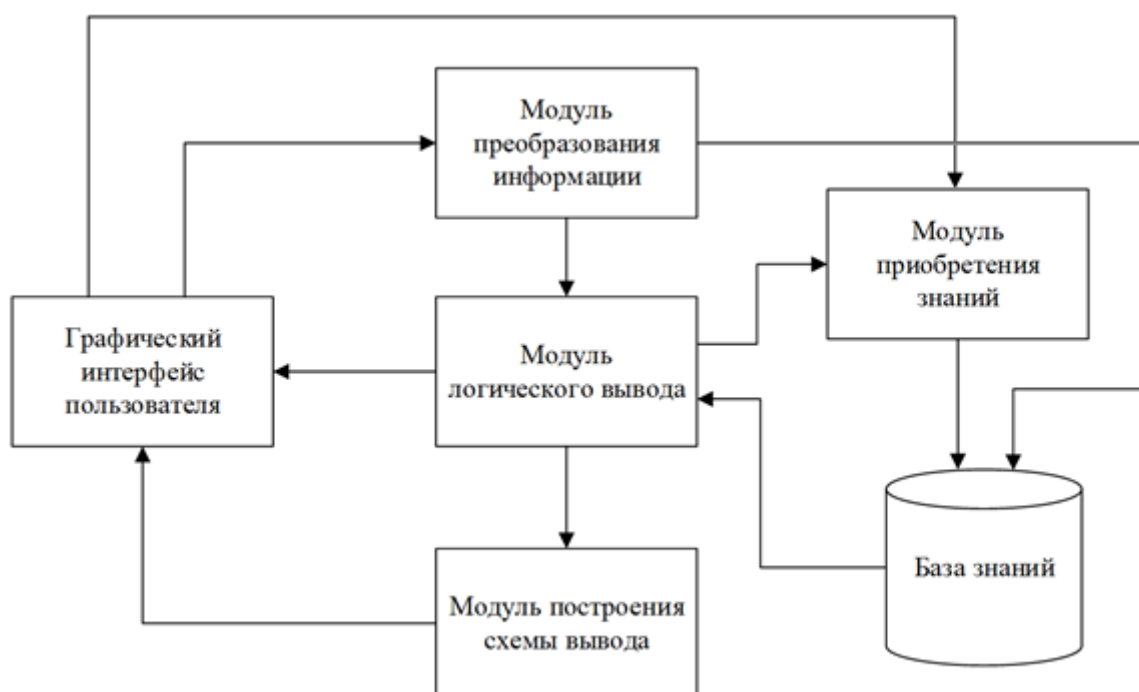


Рисунок 4.1 – Обобщенная структура программы

– модуль логического вывода определяет основные абстрактные сущности для абдуктивного и дедуктивного логического вывода, содержит реализацию данных методов и предоставляет данные о ходе вывода в графический интерфейс пользователя;

– модуль построения схемы подготавливает данные, полученные в ходе логического вывода, для графического изображения схемы в интерфейсе программы;

- база знаний хранит факты и правила предметной области, а также другие знания, необходимые для решения задач логического вывода;
- модуль преобразования знаний служит для преобразования введенной пользователем информации при составлении базы знаний в определенные сущности программы, с которыми затем осуществляется работа;
- модуль приобретения знаний реализует алгоритмы создания дополнительных правил и пополнения базы знаний на основе результатов выполнения абдуктивного логического вывода;
- графический интерфейс выполняет взаимодействие с пользователем, позволяя конфигурировать программу для работы, отображает данные о ходе решения задачи в текстовом и графическом виде.

4.2 Обзор применяемых инструментов разработки

Выбранная программная платформа Java является программным пакетом, разработанным компанией Sun Microsystems, позволяющим разрабатывать программы на одноименном языке программирования. Программный пакет для разработчиков содержит компилятор языка Java, а также богатый набор библиотек и утилит, позволяющих существенно ускорить разработку прикладных программ. Отдельно от пакета для разработчиков поставляется специализированное программное обеспечение, называемое виртуальной машиной Java. На текущий момент существует множество реализаций данной виртуальной машины для всех актуальных операционных систем, каждая из них позволяет запускать одну и ту же скомпилированную java-программу. Java используется в самых разных компьютерных платформах от встраиваемых устройств и мобильных телефонов до корпоративных серверов и суперкомпьютеров.

Программный код, написанный на языке Java преобразуется в специальный Java байт-код, который в последствии исполняется виртуальной машиной Java. Синтаксис языка в основном заимствован из таких языков программирования как С и С++, однако объектно-ориентированные возможности основаны на модели, используемой в Smalltalk и Objective-C. В Java отсутствуют определённые низкоуровневые конструкции, такие как указатели, также Java имеет очень простую модель памяти, где каждый объект расположен в куче и все переменные объектного типа являются ссылками. Управление памятью осуществляется с помощью интегрированной автоматической сборки мусора, которую выполняет JVM.

Java платформа в версии 1.5 получила масштабное обновление встроенной библиотеки многопоточности `java.util.concurrent`. Такие интерфейсы как

| | | | | | | | |
|---------------|----------------|------|----------|---------|------|--------------------|------|
| Инф. № прокл. | Подпись и дата | | | | | | |
| | Инф. № дубл. | | | | | | |
| | Взам. инф. № | | | | | | |
| | Подпись и дата | | | | | | |
| | Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | Лист |
| | | | | | | | 31 |

Программные библиотеки, разработанные на языке Java, могут быть упакованы в специальный .jar-архив для подключения к другим проектам в виде одного-единственного файла. Таким образом, более крупные проекты могут «собираться» из множества мелких библиотек, что и непосредственно отражает модульный принцип разработки ПО.

- создание шаблона и обработка ресурсов (archetype), разрешение всех зависимостей проекта;
- компиляция исходных файлов проекта;
- обработка тестовых ресурсов;
- компиляция тестов;
- тестирование;
- упаковка (создание .jar-файла)
- установка проекта в локальном Maven-репозитории (модуль становится доступным для других локальных проектов);
- установка проекта в удаленном Maven-репозитории при необходимости.

Таким образом, соответствие структуры проекта программной библиотеки стандартам Maven позволит без лишних усилий подключать разрабатываемую библиотеку к другим проектам Maven (либо подключить собранный .jar-архив), а также самой использовать Maven-библиотеки из репозиториев, доступных в сети.

Для разработки пользовательского интерфейса программы необходимо детально рассмотреть существующие библиотеки и утилиты для Java. На начальных этапах развития библиотек для разработки пользовательских интерфейсов на Java отмечались такие существенные проблемы, как неприемлемо медленная работа компонентов интерфейса и их скудная функциональность, а также чрезмерное использование ресурсов системы. Появление платформы .NET для разработки приложений с удобным и быстрым пользовательским интерфейсом под наиболее популярную операционную систему Windows еще сильнее поставило под сомнение целесообразность использования языка программирования Java и сопутствующих библиотек для создания пользовательских интерфейсов. Однако, стоит отметить, что данные проблемы не помешали Java разработчикам продолжить развивать свои продукты, которые в конечном итоге смогли составить достойную конкуренцию распространенным библиотекам.

AWT (Abstract Window Toolkit) – первая из известных библиотек разработки пользовательских интерфейсов для Java, созданная компанией Sun. Библиотека представляла из себя набор функций, которые использовали графические компоненты операционной системы. Но, для того чтобы обеспечить кроссплатформенность, разработчикам пришлось урезать функциональность многих компонентов, тем самым обеспечить единый программный интерфейс библиотеки. Таким образом к достоинствам данной библиотеки можно отнести то, что графические компоненты приложения выглядят так, как они задуманы в используемой операционной системе и при этом обеспечивается приемлемая скорость работы. Существенными недостатками является малое количество доступных стандартных графических компонентов (только доступные для ОС), а также, то, что программа, запущенная для другой ОС может выглядеть немного иначе, в большинстве случаев с внешними дефектами. На сегодняшний день библиотека AWT практически не используется.

Библиотека Swing – следующее поколение инструментов разработки пользовательских интерфейсов для Java от компании Sun. В отличие от AWT, компоненты интерфейса для Swing полностью реализованы на языке Java. Для отрисовки компонентов используется встроенный в Java платформу механизм 2D-рендеринга. Это позволило существенно расширить стандартный набор компонентов, а во-вторых облегчило разработку собственных компонентов графического интерфейса. Однако, вместе с этим, первые версии Swing работали крайне медленно, неправильно написанное приложение могло запросто вызвать нарушение работы систем семейства Windows, пользовательский интерфейс которых в данном случае переставал отвечать.

Тем не менее, библиотека Swing получила достаточно широкое распространение благодаря своей простоте, наличию большого количества

| | | | | | | | | | | | | | | |
|--|-----------------|--------------|--------------|----------------|------|------|----------|---------|------|--|--|--|--|--|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| <table border="1"> <tr> <td>Изм.</td> <td>Лист</td> <td>№ докум.</td> <td>Подпись</td> <td>Дата</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table> | | | | | Изм. | Лист | № докум. | Подпись | Дата | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| <div style="text-align: right;"> <div>ТПЖА.230101.016 ПЗ</div> <div>Лист 33</div> </div> | | | | | | | | | | | | | | |

документации и различных сообществ. Для Swing написано большое количество расширений, визуальных редакторов и других инструментов. Но и по сей день, узким местом данной библиотеки остается медленная отрисовка большого количества компонентов в окне, что, соответственно, ухудшает отзывчивость интерфейса программы. Несмотря на это, Swing является самой популярной библиотекой графических компонентов для интерфейсов программ на языке Java.

JavaFX – одна из последних разработок компании Sun, которая в очередной раз решила пересмотреть концепцию разработки пользовательских интерфейсов на Java. Данная библиотека стала существенным прорывом, здесь в отличие от AWT и Swing используется аппаратное ускорение отрисовки графических элементов. Использование лишь незначительной доли мощности графического конвейера видеокарты позволило разгрузить центральный процессор от задачи рендеринга пользовательского интерфейса, что в свою очередь не оставляет сомнений в отзывчивости даже огромного количества компонентов в окне. Такой подход позволил создать помимо стандартных элементов интерфейса также и такие сложные как компонент для просмотра трехмерных графиков и различные компоненты для работы с мультимедиа.

Начиная с Java 8, библиотека JavaFX включена в стандартный состав платформы Java и позиционируется Sun как предпочтительная для разработки пользовательских интерфейсов. Приложения, написанные с использованием данной библиотеки, будут выглядеть одинаково на всех поддерживаемых операционных системах. При этом есть возможность стилизовать приложение с помощью средств CSS, широко используемых для web-разработки.

Разработка приложений на JavaFX осуществляется согласно распространенной схеме MVC – «модель-представление-контроллер». Представление JavaFX описывается декларативно с помощью языка разметки FXML, специальной модификации XML. Пример описания простейшего представления показан на рисунке 4.2. Результат отображения представлен на рисунке 4.3.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="200.0"
    prefWidth="300.0" xmlns="http://javafx.com/javafx/8">
    <Label layoutX="106.0" layoutY="14.0" text="Click the button!"/>
    <Button layoutX="118.0" layoutY="88.0" mnemonicParsing="false" text="Click me!"/>
</AnchorPane>
```

Рисунок 4.2 – Описание представления в виде FXML-файла

| | | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|------|------|----------|---------|------|--------------------|----|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | 34 |

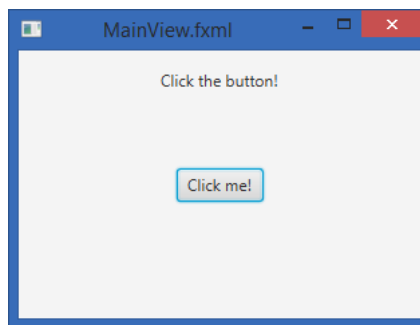


Рисунок 4.3 – Результат отображения FXML-файла

Для более удобного проектирования интерфейсов для JavaFX компанией Sun было разработано специальное приложение JavaFX Scene Builder. Оно позволяет сгенерировать FXML файл по разработанным макетам форм. Интерфейс приложения представлен на рисунке 4.4.

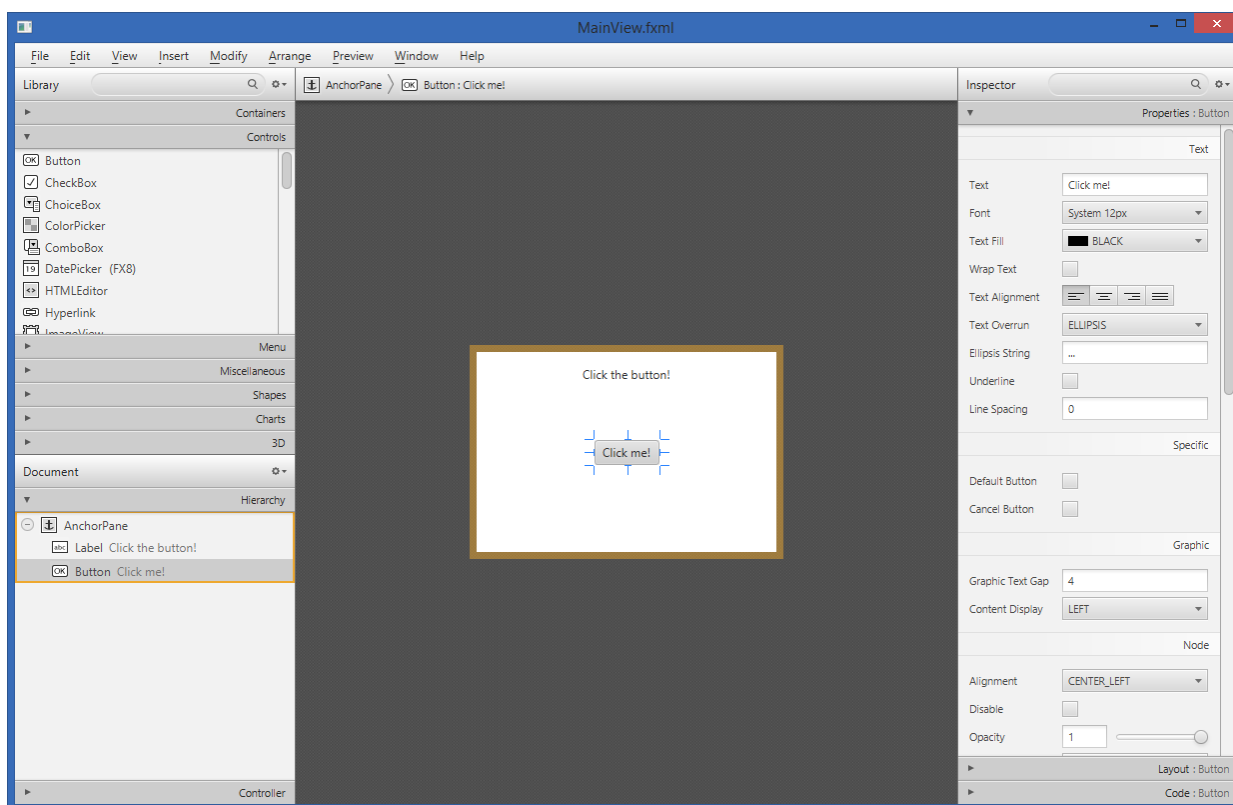


Рисунок 4.4 – Интерфейс основного окна приложения JavaFX Scene Builder

Как видно из рисунка, центральная часть интерфейса программы выделена под разрабатываемый макет. Слева сверху расположены панели выбора компонентов JavaFX, которые добавляются на макет простым перетаскиванием мышкой. Слева внизу расположена панель, отображающая текущую иерархию компонентов разрабатываемого макета для более удобной навигации. Справа в интерфейсе расположен блок настроек выделенного на макете компонента. Здесь можно настроить внешний вид компонента (раздел properties), его расположение

| | |
|----------------|----------------|
| Инв. № дубл. | Подпись и дата |
| Взам. инв. № | |
| Инв. № прошл. | |
| Подпись и дата | |
| Изм. | Лист |
| № докум. | Подпись |
| Дата | |

на форме (раздел layout), а также указать функции, вызываемые при возникновении различных событий формы, таких как щелчки мышкой, перетаскивание, нажатие клавиш и другие (раздел code).

Программные функции, привязываемые к различным событиям на форме, описываются в отдельном java-классе, который является «контроллером» для данного представления. Пример такого класса представлен на рисунке 4.5.

```
public class MainViewController {
    @FXML
    private Button button;

    @FXML
    private void buttonClicked() {
        System.out.println("You just clicked the button!");
        button.setDisable(true);
    }
}
```

Рисунок 4.5 – Пример исходного кода класса контроллера

Поля и методы класса, которые будут использоваться для представления, должны быть помечены специальной аннотацией @FXML. Далее, в приложении JavaFX Scene Builder следует отметить созданный класс как контроллер для представления, а также указать fx:id для компонента (который должен быть аналогичен названию поля в классе контроллере) и обозначить реакцию на событие «On Action» по наименованию метода в контроллере. Фрагменты интерфейса для выполнения данных действий представлены на рисунке 4.6.

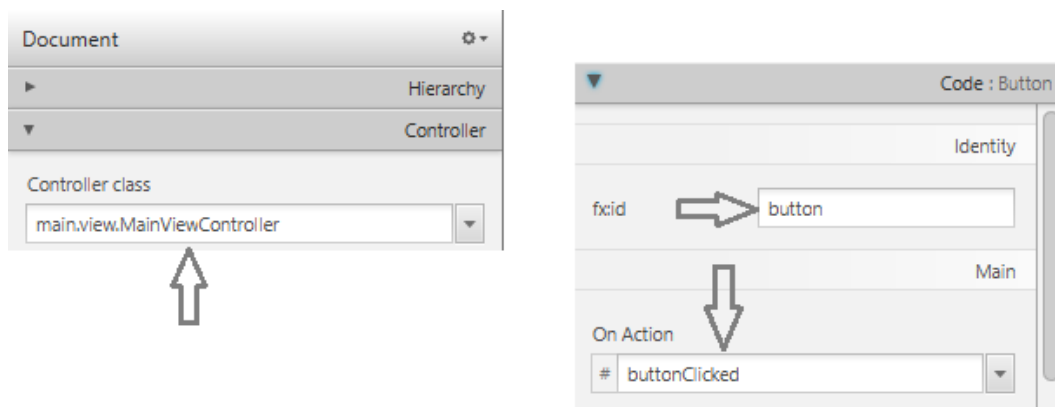


Рисунок 4.6 – Фрагменты интерфейса JavaFX Scene Builder для привязки контроллера к представлению

Таким образом, путем нескольких несложных действий, можно сконструировать приложение с графическим интерфейсом, которое может быть скомпилировано для любой поддерживаемой платформы и будет выглядеть одинаково для каждой.

| | |
|----------------|--|
| Подпись и дата | |
| Инт. № докл. | |
| Взам. инт. № | |
| Подпись и дата | |
| Инт. № прот. | |

| | | | | |
|------|------|----------|---------|------|
| | | | | |
| Изм. | Лист | № докум. | Подпись | Дата |

4.3 Разработка внутреннего представления базы знаний

Атомарной сущностью для логического вывода в логике высказываний является литерал. Литерал является элементарной формулой, которая может принимать два значения: ложь и истина (0 и 1). В большинстве случаев принято обозначать литерал заглавными или строчными буквами латинского алфавита. Также литерал может иметь интерпретацию – высказывание, записанное на естественном языке в виде повествовательного предложения.

Другим немаловажным элементом логического вывода в логике высказываний является такая сущность как «дизъюнкт» – дизъюнкция некоторого конечного числа литералов. В программном коде дизъюнкт может быть представлен как коллекция литералов, имеющая некоторые вспомогательные методы.

Таким образом после анализа основных сущностей были разработаны два класса: Literal и Disjunct, диаграмма которых представлена на рисунке 4.7. Класс Literal имеет четыре поля:

- уникальный идентификатор (id) строкового типа, позволяющий однозначно идентифицировать литерал в базе знаний;
- leftParameter и rightParameter целочисленного типа, являющиеся вспомогательными параметрами для построения схемы вывода;
- positive булевого типа, который является флагом, указывающим на то каким является литерал – положительным или отрицательным.

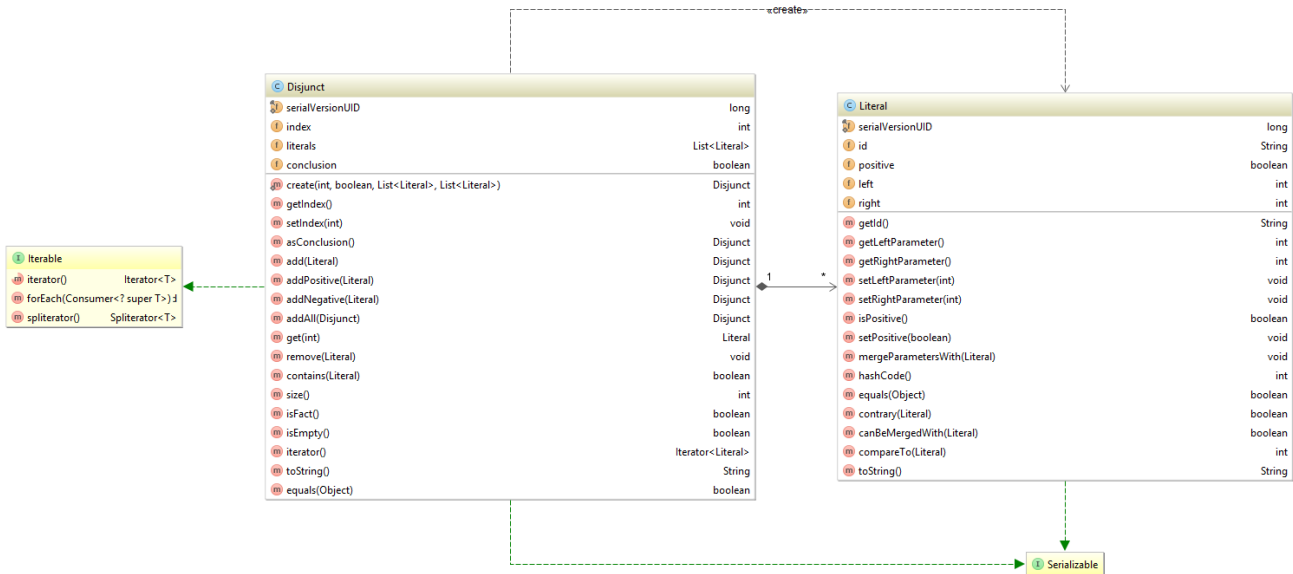


Рисунок 4.7 – Диаграмма классов Literal и Disjunct

Класс Literal имеет два конструктора, один из конструкторов принимает на вход аргумент id типа String, с которым инициализируется литерал, второй

| | |
|----------------|--|
| Подпись и дата | |
| Инв. № дубл. | |
| Взам. инв. № | |
| Подпись и дата | |
| Инв. № прошл. | |

| | | | | |
|------|------|----------|---------|------|
| | | | | |
| Изм. | Лист | № докум. | Подпись | Дата |

Данный класс в своем конструкторе инкапсулирует алгоритм деления дизъюнктов и предоставляет два статических метода для получения результата деления. Один из них, метод divide(Disjunct, Disjunct) принимает на вход два аргумента – дизъюнкт-делимое и дизъюнкт-делитель соответственно, и возвращает в качестве результата объект класса DisjunctDivision.

Объекты класса DisjunctDivision содержат два поля – remainder (остаток) и quotient (частное), готовых к дальнейшей обработке алгоритмом логического вывода. Второй статический метод данного класса groupDivision(Disjunct, Disjunct) предназначен для выполнения операции группового деления дизъюнктов, и он необходим при реализации алгоритма абдуктивного логического вывода для формирования множества дополнительных посылок.

Конструктор класса DisjunctDivision объявлен как private, что говорит об ограничении создания экземпляров данного класса. Данное решение позволяет объединить операции деления дизъюнктов в один смысловой блок и использовать так называемые «фабричные» методы класса для создания экземпляров и выполнения алгоритмов деления дизъюнктов.

4.4.2 Дерево логического вывода

В описании задачи логического вывода заключений с построением схемы логического вывода фигурируют такие понятия как обобщенная процедура деления, шаг вывода, дерево вывода. На основе этих сущностей были созданы соответствующие классы, такие как Procedure, InferenceTreeStage, InferenceTree, входящие в состав модуля логического вывода.

В основе метода логического вывода лежит специальная процедура вывода, названная процедурой обобщенного деления дизъюнктов. Данная процедура содержит несколько важных полей, таких как множество дизъюнктов исходных секвенций, дизъюнкт секвенции-заключения, множество остатков и множество частных деления дизъюнктов, множества новых исходных и выводимых секвенций, а также состояние вывода. Абстрактный класс Procedure содержит в себе все перечисленные поля и является неким обобщением для процедур дедуктивного и абдуктивного логического вывода. Классы-наследники Procedure должны предоставить реализацию фабричного метода newInstance, который возвращает новый экземпляр подкласса процедуры. Это позволяет абстрагироваться от использования конкретных процедур вывода и сделать класс InferenceTreeStage (шаг дерева вывода) универсальным и способным работать с любым экземпляром класса-наследника Procedure. На рисунках 4.12 и 4.13 продемонстрирован пример объявления метода newInstance в классе Procedure и его реализации классом DeductiveProcedure.

| | | | | | | | | | | |
|----------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инф. № прораб. | Подпись и дата. | Взам. инб. № | Инб. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 40 |

Кроме того, класс Procedure реализует интерфейс Callable<T>, описанный в пакете java.util.concurrent, содержащий объявление метода call, предназначенного для инкапсуляции задачи, которую необходимо будет выполнить асинхронно, в отдельном потоке. И зачастую необходимо, чтобы поток, после выполнения своей работы возвращал некоторое значение. Метод call является шаблонным и может возвращать значение типа T, указанного в качестве generic-параметра для интерфейса.

```
public abstract class Procedure implements Callable<State>, Serializable {

    /**
     * Поля класса
     * ...
     */

    public abstract Procedure newInstance(List<Disjunct> source, Disjunct conclusion);

    /**
     * Другие методы класса
     * ...
     */
}
```

Рисунок 4.12 – Фрагмент кода объявления фабричного метода newInstance

```
public class DeductiveProcedure extends Procedure {
    public DeductiveProcedure(List<Disjunct> source, Disjunct conclusion) {
        super(source, conclusion);
    }

    @Override
    public Procedure newInstance(List<Disjunct> source, Disjunct conclusion) {
        return new DeductiveProcedure(source, conclusion);
    }
}
```

Рисунок 4.13 – Фрагмент кода реализации класса DeductiveProcedure

В случае класса Procedure метод call возвращает состояние процедуры (State), который в свою очередь объявлен как класс-перечисление (enum). Конструкция Enum позволяет ограничить множество допустимых значений для некоторого типа данных и оперировать ими с помощью произвольно заданных строковых констант. В данном случае перечисление State содержит три возможных значения, отражающих возможные состояния процедуры в

| | | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|------|------|----------|---------|------|--------------------|----|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | 41 |

определенный момент времени: INCOMPLETE (незавершена), SUCCESSFUL (успешно завершена) и FAILED (завершена неудачно).

Рассмотрев алгоритмы дедуктивного и абдуктивного логического вывода, можно заметить, что они имеют общий порядок выполнения операций, лишь с незначительными отличиями. Такие операции как деление исходных дизъюнктов, анализ остатков, деление остатков на факты, конъюнкция остатков могут быть реализованы в абстрактном классе единой. Данный принцип отражает такой прием проектирования как «шаблонный метод», применяемый в случаях, когда:

- возможно однократно определить последовательность частей алгоритма и оставить реализацию этих частей подклассам;
- необходимо инкапсулировать в одном классе общее поведение для всех подклассов, избежав тем самым дублирования кода.

Таким образом метод call для класса Procedure разбивается на три этапа: деление исходных дизъюнктов, деление остатков на дизъюнкт и конъюнкция остатков. Каждый этап реализует отдельный protected метод, доступный для перегрузки в классах-потомках. Исходный код метода call класса Procedure представлен на рисунке 4.14.

```
@Override
public State call() throws Exception {
    // Деление исходных дизъюнктов
    divideSourceDisjuncts();

    // Один из остатков после деления = 0. Процедура была завершена успешно.
    if (state.equals(State.SUCCESSFUL)) {
        return state;
    }

    // Остатки отсутствуют. Процедура завершена неудачно.
    if (remainders.isEmpty()) {
        state = State.FAILED;
        return state;
    }

    // Деление остатков на дизъюнкт, составленный из фактов
    divideRemaindersByFacts();

    // Один из остатков после деления = 0. Процедура была завершена успешно.
    if (state.equals(State.SUCCESSFUL)) {
        return state;
    }

    // Конъюнкция остатков.
    remaindersConjunction();

    return state;
}
```

Рисунок 4.14 – Исходный код метода call класса Procedure

| | | | | | | |
|--|------|------|----------|---------|------|-------------------------------|
| Инв. № прокл. | Изм. | Лист | № докум. | Подпись | Дата | Подпись и дата |
| | | | | | | Инв. № дубл. |
| | | | | | | Взам. инв. № |
| | | | | | | Подпись и дата |
| <div style="text-align: right;">ТПЖА.230101.016 ПЗ</div> | | | | | | <div>Лист</div> <div>42</div> |

В алгоритме абдуктивного метода после формирования конъюнкции остатков предусмотрена специальная операция группового деления дизъюнктов. Таким образом, достаточно лишь перегрузить protected метод remaindersConjunction() класса Procedure в новом классе-потомке AbductiveProcedure и добавить код для новой операции. Пример перегрузки метода представлен на рисунке 4.15.

```

@Override
protected void remaindersConjunction() {
    super.remaindersConjunction();

    this.addition = DisjunctDivision.groupDivision(conclusion, dividedDisjunctLiterals);
}
    
```

Рисунок 4.15 – Пример перегрузки метода remaindersConjunction

Диаграмма класса Procedure и его наследных классов представлена на рисунке 4.16.

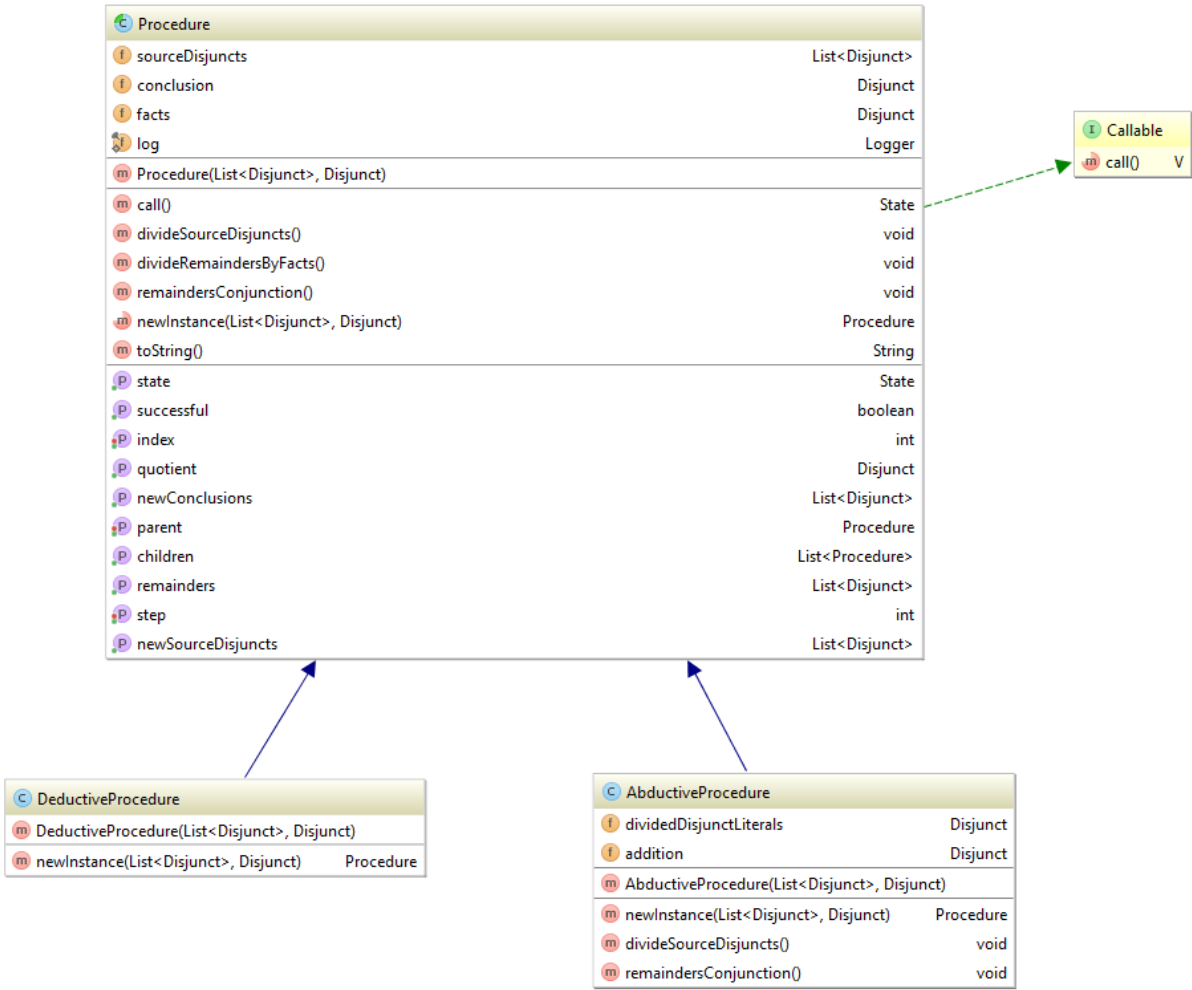


Рисунок 4.16 – Диаграмма класса Procedure и его наследников

| | | |
|--------------|----------------|--|
| Инв. № дубл. | Подпись и дата | |
| | | |
| Взам. инв. № | Подпись и дата | |
| | | |
| Инв. № прот. | Подпись и дата | |
| | | |

– пул создает объекты класса Thread по мере необходимости, то есть если все потоки в сервисе заняты выполнением задач, то для новой задачи создастся новый поток;

– пул повторно использует неактивные потоки, исключая лишние вызовы оператора new;

– пул подчищает потоки, которые были неактивны некоторый продолжительный промежуток времени, тем самым освобождая память после выполнения большого числа потоков.

Таким образом имеет смысл использовать один экземпляр ExecutorService для всех шагов алгоритма.

Объект класса InferenceTreeStage инициализируется тремя параметрами: номер шага, исходные процедуры и сервис запуска потоков. Поскольку объекты класса Procedure реализуют интерфейс Callable, они могут быть переданы в сервис на выполнение. Также предусмотрено сохранение указателей на объекты класса Future – результатов выполнения процедур для последующего анализа. Результаты записываются в структуру данных HashMap, элементы которой являются парами «ключ-значение», ключом в данном случае выступает указатель на процедуру, а значением – результат её выполнения. После передачи всех процедур на выполнение выполняется обход списка результатов, в каждой итерации которого вызывается метод get на объекте класса Future (результат выполнения процедуры). Таким образом реализуется ожидание выполнения процедур шага. Далее анализируются результаты выполнения и, если вывод еще не завершен, создаются новые процедуры для следующего шага.

Схема процесса выполнения параллельного логического вывода представлена на рисунке 4.18.

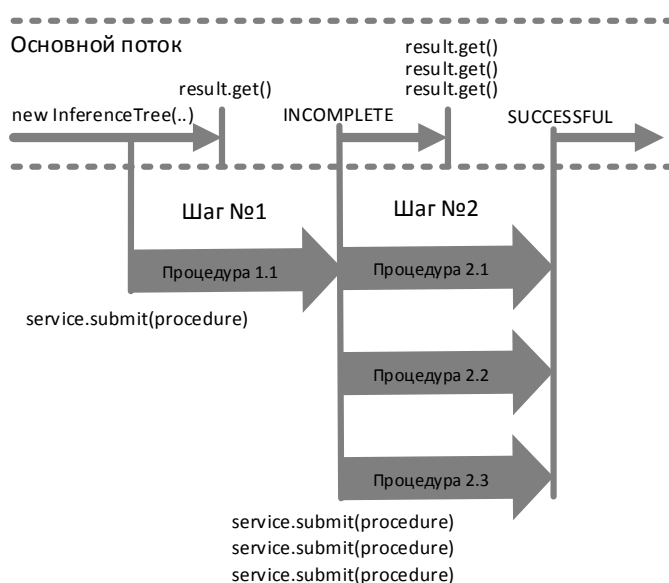


Рисунок 4.18 – Схема процесса выполнения параллельного логического вывода

| | |
|----------------|----------------|
| Инв. № дубл. | Подпись и дата |
| Взам. инв. № | |
| Подпись и дата | |
| Инв. № продл. | |

| | | | | |
|------|------|----------|---------|------|
| Изм. | Лист | № докум. | Подпись | Дата |
|------|------|----------|---------|------|

Класс InferenceTree является реализацией дерева логического вывода, его конструктор принимает на вход объект абстрактного класса Procedure, а уже конкретный класс данного объекта определяет вид процедуры, а значит и вид дерева. Объект класса InferenceTree создает экземпляры класса InferenceTreeStage, пока один из них не завершится со статусом (State) успешно (SUCCESSFUL) или неуспешно (FAILED). Диаграмма классов InferenceTreeStage и InferenceTree представлена на рисунке 4.19.

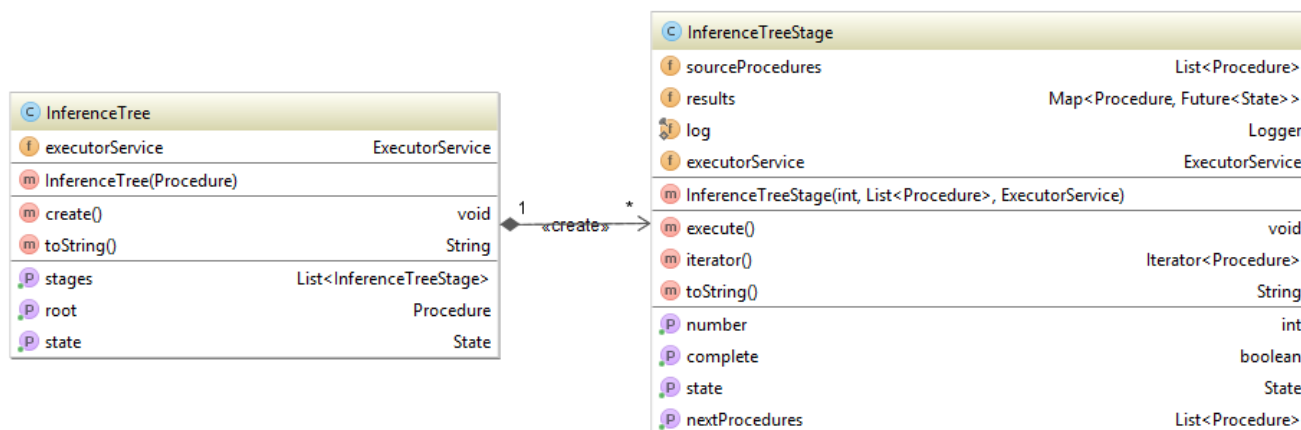


Рисунок 4.19 – Диаграмма классов InferenceTreeStage и InferenceTree

После завершения выполнения логического вывода полученное дерево (объект класса InferenceTree) может быть проанализировано путем обхода «сначала вглубь» с помощью корневой процедуры (поле root) и рекурсивного обхода всех дочерних процедур, а также «сначала вширь» с помощью итераций по списку шагов дерева.

Полученная структура классов в полной мере реализует методы дедуктивного и абдуктивного логического вывода, созданные объекты содержат информацию о ходе вывода. Например, частные, полученные в ходе процедур деления дизъюнктов используются для построения схемы логического вывода, в случае если вывод завершен удачно. А также дополнения, полученные после группового деления дизъюнктов в каждой процедуре абдуктивного вывода, используются для формирования формул дополнительных посылок.

4.4.3 Журналирование данных логического вывода

Важным требованием при разработке различных программных библиотек или «фреймворков» является обеспечение журналирования отладочной информации, а также сведений о возникающих ошибках. Также используется такое понятие как «логирование» информации, или ведение «логов». В настоящее время практически невозможно найти какое-либо приложение, поддерживаемое

| | |
|----------------|----------------|
| Инф. № прораб. | Подпись и дата |
| Взам. инф. № | Инф. № дубл. |
| Инф. № дубл. | Подпись и дата |
| Инф. № прораб. | Подпись и дата |

| | | | | | | |
|------|------|----------|---------|------|--------------------|------|
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | Лист |
| | | | | | | 46 |

разработчиками, без использования логирования. Самым простым и широко распространенным способом ведения логов – сохранение их в текстовый файл. При этом каждое отдельное событие представляет собой отдельную строку.

Платформа Java позволяет использовать множество способов ведения логов, от простой печати в консоль методом `System.out.println()` до более продвинутого встроенного пакета `java.util.logging`. Также следует обратить внимание на библиотеки логирования от сторонних разработчиков, такие как Logback и log4j. Последняя недавно получила крупное обновление, были добавлены механизмы асинхронного ведения логов, использование которых более предпочтительно для многопоточных программ. Подключение библиотеки log4j для Maven проектов осуществляется лишь подключением зависимости в `pom.xml` файле.

Библиотека log4j предоставляет возможность гибкой конфигурации ведения логов. С помощью файла настроек (XML или JSON) можно определить формат вывода сообщений, стратегию вывода сообщений, указать тип файла, максимальный размер и так далее. Для асинхронного логирования свойственна стратегия отложенной записи файл.

В библиотеке log4j заданы следующие уровни сообщений: `off > fatal > error > warn > info > debug > trace > all`. Для разработки и отладки предпочтительно использование `debug`-сообщений, для конечного продукта сообщений уровня `error`. С помощью настроек можно задать максимальный уровень выводимых сообщений. Таким образом есть возможность отключить вывод отладочной информации и повысить производительность заверщенного приложения.

Шаблон сообщений в настройках определяет формат вывода информации в лог. Возможен вывод системного времени, уровня сообщения, идентификатора потока, а также любой пользовательский текст. Фрагмент отладочного журнала для библиотеки логического вывода представлен на рисунке 4.20.

```
2014-12-15 15:05:15,635 DEBUG [pool-3-thread-4] - 2.4: процедура незавершена (необходимо продолжение решения)
2014-12-15 15:05:15,635 DEBUG [main] - * Шаг №2 завершен.
2014-12-15 15:05:15,635 DEBUG [main] - * Завершенных процедур: 3 из 4. Неудачно завершенных процедур: 0
2014-12-15 15:05:15,635 DEBUG [main] - * Количество порожденных процедур: 16

2014-12-15 15:05:15,635 DEBUG [main] - * Инициализация шага №3. Количество выполняемых процедур: 16
2014-12-15 15:05:15,635 DEBUG [pool-3-thread-4] - 3.1: выполнение процедуры запущено
2014-12-15 15:05:15,635 DEBUG [pool-3-thread-2] - 3.2: выполнение процедуры запущено
2014-12-15 15:05:15,635 DEBUG [pool-3-thread-3] - 3.3: выполнение процедуры запущено
2014-12-15 15:05:15,635 DEBUG [pool-3-thread-4] - 3.1: процедура завершена успешно (один из остатков = 0)
2014-12-15 15:05:15,635 DEBUG [pool-3-thread-4] - 3.15: выполнение процедуры запущено
2014-12-15 15:05:15,635 DEBUG [pool-3-thread-1] - 3.4: выполнение процедуры запущено
2014-12-15 15:05:15,636 DEBUG [pool-3-thread-3] - 3.3: процедура завершена успешно (один из остатков = 0)
```

Рисунок 4.20 – Фрагмент отладочного журнала библиотеки

| | | | | | | | | | |
|--------------|-----------------|--------------|--------------|-----------------|--------------------|--|--|--|--|
| Инф. № прот. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата. | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | |
| | | | | | 47 | | | | |

4.5 Разработка программной библиотеки преобразования информации в формат базы знаний

Описание базы знаний сводится к вводу набора правил в текстовом виде для последующего преобразования их в дизъюнкты. Удобнее всего вводить правила в виде импликаций – логических связей вида «если...то...». Для реализации механизма перевода исходных правил в набор дизъюнктов необходимо разработать специальный язык, позволяющий описать базу знаний в удобочитаемом виде. Для этого потребуется составление грамматики языка с помощью формальной системы определения синтаксиса. Далее необходимо разработать лексический и синтаксический анализаторы, совместная работа которых позволит преобразовать исходный поток символов в набор дизъюнктов, учитывая при этом возможные ошибки ввода.

4.5.1 Описание грамматики языка

Грамматика большинства языков программирования является контекстно-свободной (тип 2 иерархии Хомского), которая может быть описана с помощью формальной системы описания синтаксиса – формой Бэкуса-Наура (БНФ).

Контекстно-свободная грамматика содержит четыре вида компонентов:

- множество терминалов (или терминальных символов), представляющие собой атомарные символы языка, которые определяются грамматикой;
- множество нетерминалов (или нетерминальных символов), представляющие собой множество строк терминалов, описанных тем или иным способом;
- множество продукционных правил, каждое из которых состоит из левой и правой части; левой частью правила является нетерминал, а правой может быть последовательность терминалов и/или нетерминалов;
- нетерминальный символ, который является начальным для грамматики.

Грамматика определяется набором продукционных правил, причем первым правилом задается начальный символ.

Для ввода исходных правил разработана следующая грамматика языка, представленная на рисунке 4.21.

| | | | | | | | | | |
|----------------|-----------------|--------------|--------------|-----------------|--------------------|--|--|--|------|
| Инв. № прораб. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата. | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | Лист |
| | | | | | | | | | 48 |


```

LinceParser = {Sequence} .
Sequence = ['if'] Antecedent ('then' | '>') Consequent ';' .
Consequent = 'false' | '0' | OrExpression .
Antecedent = 'true' | '1' | AndExpression .
OrExpression = Literal {'or' | '|'} Literal .
AndExpression = Literal {'and' | '&'} Literal .
Literal = lit | '~' lit .
lit = letter {letter | digit | '_'} .
letter = 'A' ... 'Z' + 'a' ... 'z' .
digit = '0' ... '9' .

```

Рисунок 4.21 – Грамматика языка в РБНФ

Грамматика задана с помощью РБНФ-описания (расширенная форма Бэкуса-Наура), которое позволяет описать более емкие конструкции языка, по сравнению с классической формой. Набор конструкции описывается следующим образом:

- лексема «=» ее описание;
- $\{A\}$ – 0 или более элементов A ;
- $[A]$ – 0 или 1 элементов A ;
- $(A\ B)$ – группировка элементов;
- $A\ |\ B$ – элемент A или элемент B ;
- $A\ B$ – элемент A , за которым следует элемент B ;
- ... – группа символов «от» и «до».

Начальным символом здесь является нетерминал LinceParser, продукционное правило для которого задает последовательность нетерминалов Sequence, заканчивающихся символом «;». Фигурные скобки в данной записи означают «ноль и более элементов». Нетерминал Sequence в свою очередь определяется как необязательный символ «if», нетерминал Antecedent, далее символ «then» или «>» и нетерминал Consequent. Antecedent и Consequent могут содержать символы 1 и 0 соответственно (или true и false), а также нетерминальные символы AndExpression и OrExpression соответственно. AndExpression и OrExpression являются конъюнкцией и дизъюнкцией литералов, разделяемых символами «and» и «or» (или «&» и «|»). Literal может быть представлен как положительным, так и отрицательным с символом «~». Literal может быть записан как последовательность букв латинского алфавита, цифр и символа « », однако начинаться может только с буквы.

Пример записи правил для базы знаний с помощью разработанной грамматики представлен на рисунке 4.22.

```

if A and B and C then D or E;
if F and G then H;
if true then J;

A & B & C > D | E;
F & G > H;
1 > J;

```

Рисунок 4.22 – Пример записи правил для базы знаний

Первые три правила, представленные на рисунке 4.22 записаны с помощью ключевых слов «if», «and», «then» и «or». Данный вариант выглядит более предпочтительным, поскольку он схож с правилами, записанными на естественном языке. Замена ключевых слов на символы, как в последних трех правилах на рисунке 4.22, допустима и предполагается для того, чтобы загрузить и считать правила из текстового файла, который предназначался для программы, описанной в разделе 1.2, при минимальном количестве требуемых изменений. Также возможно комбинировать ключевые слова и их символьные аналоги в рамках одного или нескольких правил.

4.5.2 Лексический анализатор

Лексический анализ представляет собой первую фазу разбора входного потока данных и его основная задача – чтение входных символов и группировка их в токены, которые в дальнейшем передаются синтаксическому анализатору.

В большинстве случаев лексический анализатор представляется в виде конечного автомата, в котором кодируется информация о возможных последовательностях символов. При этом, как правило, первый символ, отличный от пробельного, определяет начало следующего токена, после чего последующие считанные символы записываются в буфер до тех пор, пока не встретится символ, недопустимый для текущего токена или следующий пробельный символ. После этого текущая строка, хранящаяся в буфере, представляет собой необработанный токен, для которого необходимо определить тип с помощью регулярных выражений или вычисления ключевого значения по символам.

В настоящее время разработка лексического анализатора не является чем-то уникальным для той или иной задачи. Разработано множество алгоритмов и программных средств для генерации лексических анализаторов, которые различаются используемыми языками программирования и способом описания токенов. Все эти программные средства позволяют избавиться от рутинных задач,

| | | | | | | |
|---------------|----------------|------|----------|---------|--------------------|------|
| Инв. № прокл. | Подпись и дата | | | | ТПЖА.230101.016 ПЗ | Лист |
| | Инв. № дубл. | | | | | 50 |
| | Взам. инв. № | | | | | |
| | Подпись и дата | | | | | |
| | Изм. | Лист | № докум. | Подпись | Дата | |

| | | | | |
|---------------|-----------------|-------------|--------------|----------------|
| Инв. № прокл. | Подпись и дата. | Взам. инв № | Инв. № дубл. | Подпись и дата |
| | | | | |

```
CHARACTERS
letter = 'A' .. 'Z' + 'a' .. 'z' .
digit = '0' .. '9' .

TOKENS
lit = letter {letter | digit | '_' } .
if = "if" .
then = "then" .
and = "and" .
or = "or" .
neg = '~' .
true = "true" .
false = "false" .
zero = '0' .
one = '1' .

COMMENTS FROM "/*" TO "*/" NESTED
COMMENTS FROM "//" TO '\n'

IGNORE '\r' + '\n' + '\t'
```

| | |
|-----------------|--|
| Подпись и дата. | |
|-----------------|--|

| | |
|---------------|--|
| ИНВ. № продл. | |
|---------------|--|

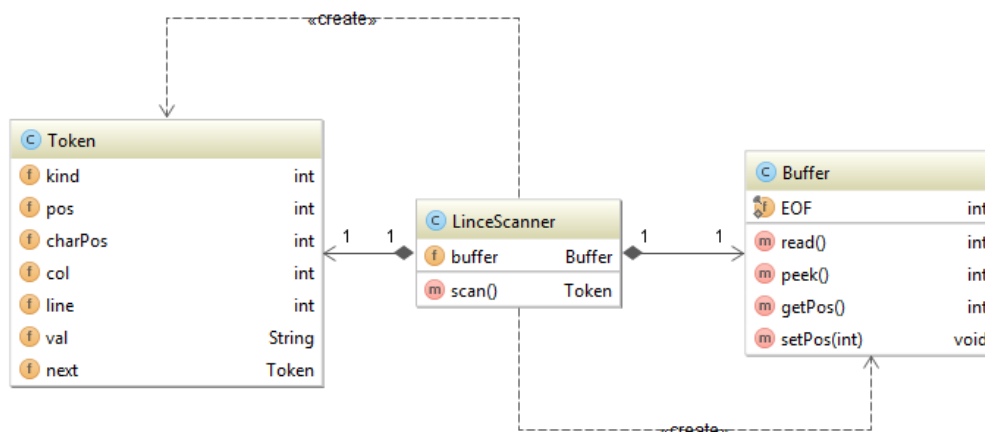


Рисунок 4.24 – Диаграмма классов лексического анализатора

Класс Token представляет собой базовую структуру данных для лексического анализатора и имеет следующие поля:

- kind – код типа токена;
- pos – позиция в потоке (в байтах);
- charPos – позиция в потоке (в символах);
- col – позиция в строке, в которой находится токен;
- line – номер строки;
- val – строковое значение токена;
- next – указатель на следующий токен в потоке.

LinceScanner содержит описание набора внутренних состояний конечного автомата, сгенерированного по описанию лексем грамматики. В каждый момент времени автомат находится в определенном состоянии, переходы между состояниями осуществляются после анализа следующего символа во входной последовательности. Также был сгенерирован специальный Buffer, предназначенный для работы с UTF-8 символами и обработки некорректных символов.

Работа сканера осуществляется с помощью метода scan, который возвращает следующий токен в потоке. Сами же токены организованы по принципу однонаправленного связного списка, каждый из них содержит указатель на следующий токен в потоке.

4.5.3 Синтаксический анализатор

Второй фазой интерпретации введенных данных является синтаксический анализ. Стандартная работа синтаксического анализатора заключается в

| | | | | |
|---------------|-----------------|--------------|--------------|----------------|
| Инф. № прокл. | Подпись и дата. | Взам. инф. № | Инф. № дубл. | Подпись и дата |
| | | | | |

| | | | | | | |
|------|------|----------|---------|------|--------------------|------|
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | Лист |
| | | | | | | 52 |

получении токенов от лексического анализатора и определении того, может ли текущая последовательность токенов порождаться грамматикой языка. При этом может быть построено дерево синтаксического разбора для последующего анализа семантики, однако для простых грамматик зачастую буквального построения дерева не требуется, поскольку все дальнейшие этапы интерпретации исходных выражений можно выполнить в рамках синтаксического анализа.

Существует несколько способов синтаксического анализа. Все они, в общем случае, отличаются по тому, работают ли они в нисходящем направлении (от начального символа грамматики, строя синтаксическое дерево сверху вниз) или в восходящем (начиная с терминальных символов, строя дерево разбора снизу вверх).

Грамматика языка, описанная в разделе 4.5.1, является однозначной (некоторая строка терминалов имеет только одно дерева разбора) без левых рекурсий (ситуация, когда нетерминальный символ порождает последовательность, которая начинается с этого же нетерминала). Такой вид грамматик называется LL(1)-грамматики. Самой простой и эффективной реализацией синтаксического разбора для данной грамматики является синтаксический анализ методом рекурсивного спуска без возврата или предиктивный синтаксический анализ. Программная реализация будет состоять из набора процедур, каждая из которых предназначена для анализа конкретного нетерминала. Алгоритм синтаксического разбора инкапсулирован в классе LinceParser, диаграмма которого представлена на рисунке 4.25.

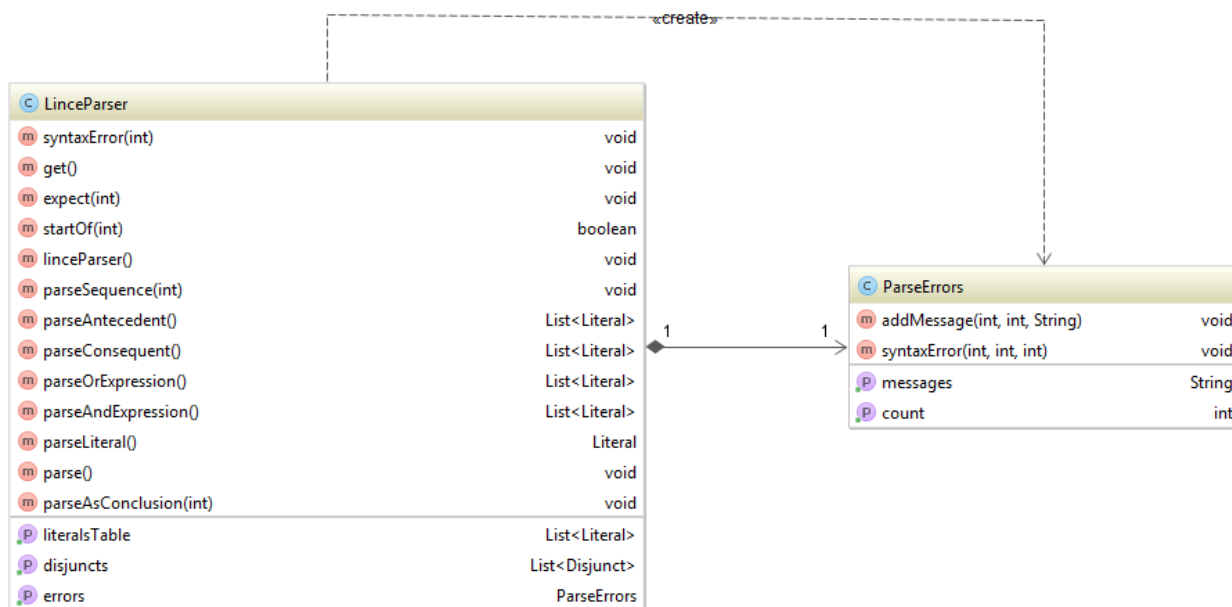


Рисунок 4.25 – Диаграмма классов LinceParser и ParseErrors

| | | | | |
|---------------|-----------------|-------------|--------------|----------------|
| Инв. № продл. | Подпись и дата. | Взам. инв № | Инв. № дубл. | Подпись и дата |
| | | | | |



| |
|------|
| Лист |
| 54 |

Класс Graph является наследником класса ScrollPane – стандартного компонента JavaFX, представляющего собой контейнер, для которого возможно настроить отображение полос прокрутки, если его содержимое выходит за рамки панели. Graph имеет поле graphContent, которое является объектом класса Group – еще одного стандартного компонента JavaFX, именно в нем размещаются вершины графа. Group автоматически размещает дочерние объекты относительно своего центра и при этом все трансформации, применяемые к Group, также применяются и к дочерним объектам. Также Graph содержит списки входящих в него вершин и дуг. Добавляются вершины и дуги с помощью методов addVertex и addEdge, в которых помимо добавления переданных объектов во внутренние коллекции осуществляется их предварительная обработка: для вершин регистрируются события мыши, а для дуг рассчитывается путь.

На рисунке 4.28 представлена диаграмма класса GraphVertex и наследного CircleVertex.

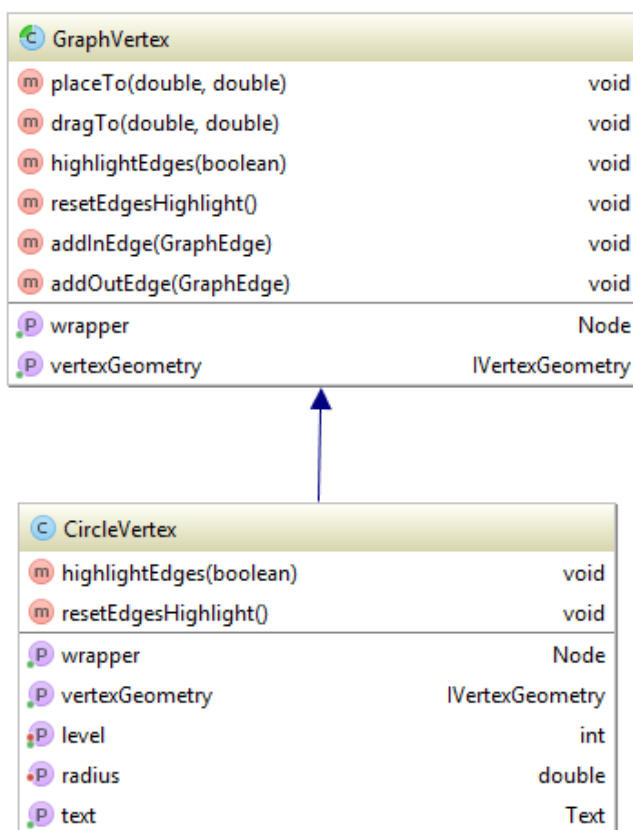


Рисунок 4.28 – Диаграмма классов GraphVertex и CircleVertex

Класс GraphVertex (вершина графа) объявлен как абстрактный. Это говорит о невозможности создания объектов данного класса. Разработчик, использующий библиотеку может представить свой подкласс GraphVertex, либо воспользоваться одной из стандартных реализаций, к примеру, CircleVertex. Поскольку

| | |
|----------------|----------------|
| Инд. № прораб. | Подпись и дата |
| Взам. инд. № | Инд. № дубл. |
| Подпись и дата | |

GraphVertex является абстрактной сущностью, а в JavaFX необходимо работать с реальными компонентами, в классе был объявлен абстрактный метод getWrapper, который должен возвращать какой-либо компонент JavaFX, наследуемый от стандартного класса Node. Подклассы GraphVertex должны предоставить реализацию этого метода. Другими словами, класс GraphVertex является оберткой для любого компонента JavaFX, предоставляя возможность изобразить граф из любых компонентов в качестве вершин. Таким образом, методы по перемещению узла dragTo и placeTo, реализованные в GraphVertex, будут вызывать метод getWrapper для непосредственной работы с компонентом узла.

Класс GraphEdge – ребро графа, содержит такие поля как начальная вершина, конечная вершина, а также объект класса Path – в JavaFX – путь вида MoveTo-LineTo. Объект класса Path является составным, к примеру, для построения ломаной линии понадобится один объект MoveTo и несколько объектов LineTo. Для построения ребра графа используется следующий алгоритм:

- получить координаты центра начальной вершины в рамках родительского элемента, т.е. панели графа;
- создать объект класса MoveTo с координатами центра начальной вершины и добавить его в Path;
- получить координаты центра конечной вершины в рамках родительского элемента;
- создать объект класса LineTo с координатами центра конечной вершины и добавить его в Path.

Таким образом, вызов метода, реализующего данный алгоритм, осуществляется при каждой смене координат начальной или конечной вершины для того чтобы перерисовать ребро.

4.6.2 Механизм построения ребер ориентированного графа

Поскольку схема логического вывода является ориентированным графом, необходимо обеспечить отрисовку направления дуги. Однако в Path не предусмотрено добавление никаких других элементов, кроме линий. Для изображения стрелки на дуге создается объект класса Polygon с тремя точками. Но для чтобы стрелка меняла свое положение и направление в зависимости от перемещения вершин необходимо применить для отрисовки некоторые формулы тригонометрии. На рисунке 4.29 представлена геометрия соприкосновения вершины и ребра графа.

| | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|
| Инв. № прадл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | |
| | | | | | 57 | | | | |

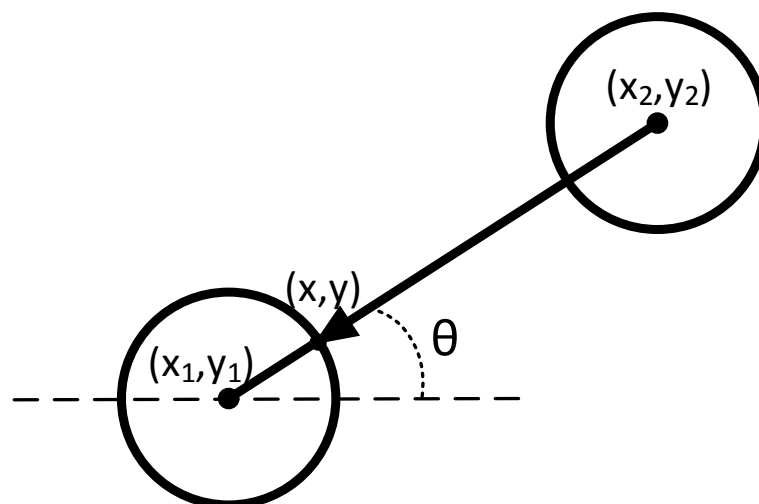


Рисунок 4.29 – Геометрия соприкосновения вершины и ребра графа

Определение угла поворота стрелки заключается в нахождении угла наклона дуги по отношению к горизонтальной оси. Для этого используется следующая формула

$$\theta = \arctg \frac{y_2 - y_1}{x_2 - x_1}, \quad (4.1)$$

где θ – угол наклона отрезка (рад);

x_1, y_1 и x_2, y_2 – координаты конечных точек отрезка.

Вдобавок, следует отметить, что по мере вращения дуги вокруг вершины, стрелка, входящая в вершину будет постоянно менять координаты точки касания с окружностью вершины. Для вычисления координат данной точки касания используются следующие формулы

$$x = x_c + \cos \theta \cdot R \quad (4.2)$$

$$y = y_c + \sin \theta \cdot R, \quad (4.3)$$

где x_c, y_c – координаты центра окружности;

θ – угол наклона отрезка;

R – радиус окружности.

Таким образом, с помощью перечисленных формул, обеспечивается корректность отрисовки стрелки на конце дуги при смене положения вершин.

Выше был рассмотрен расчет геометрии ребер графа, при котором вершины графа имеют форму окружности. Но в случае, когда вершинами графа являются объекты прямоугольной формы, данные формулы не позволят рассчитать

| | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|
| Инв. № прадл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ |
| | | | | | Лист 58 |

корректную геометрию. В рамках разработки программы логического вывода нет необходимости в отображении графа с прямоугольными вершинами, однако при повторном использовании библиотеки в других проектах может понадобиться такая возможность. Для этого предусмотрен следующий механизм. Объявлен интерфейс IVertexGeometry с методом calculateArrowPosition, принимающий на вход указатель на объект стрелки, а также координаты положения начальной и конечной вершины. Абстрактный класс GraphVertex содержит абстрактный метод getVertexGeometry, который в случае его определения потомками GraphVertex должен вернуть объект, реализующий интерфейс IVertexGeometry. К примеру, для вершины круглой формы CircleVertex создан класс CircleShapedVertexGeometry, содержащий информацию о радиусе вершины, а также реализующий расчет геометрии стрелки для вершин круглой формы в методе calculateArrowPosition. Диаграмма классов вершин и метода расчета геометрии представлена на рисунке 4.30.

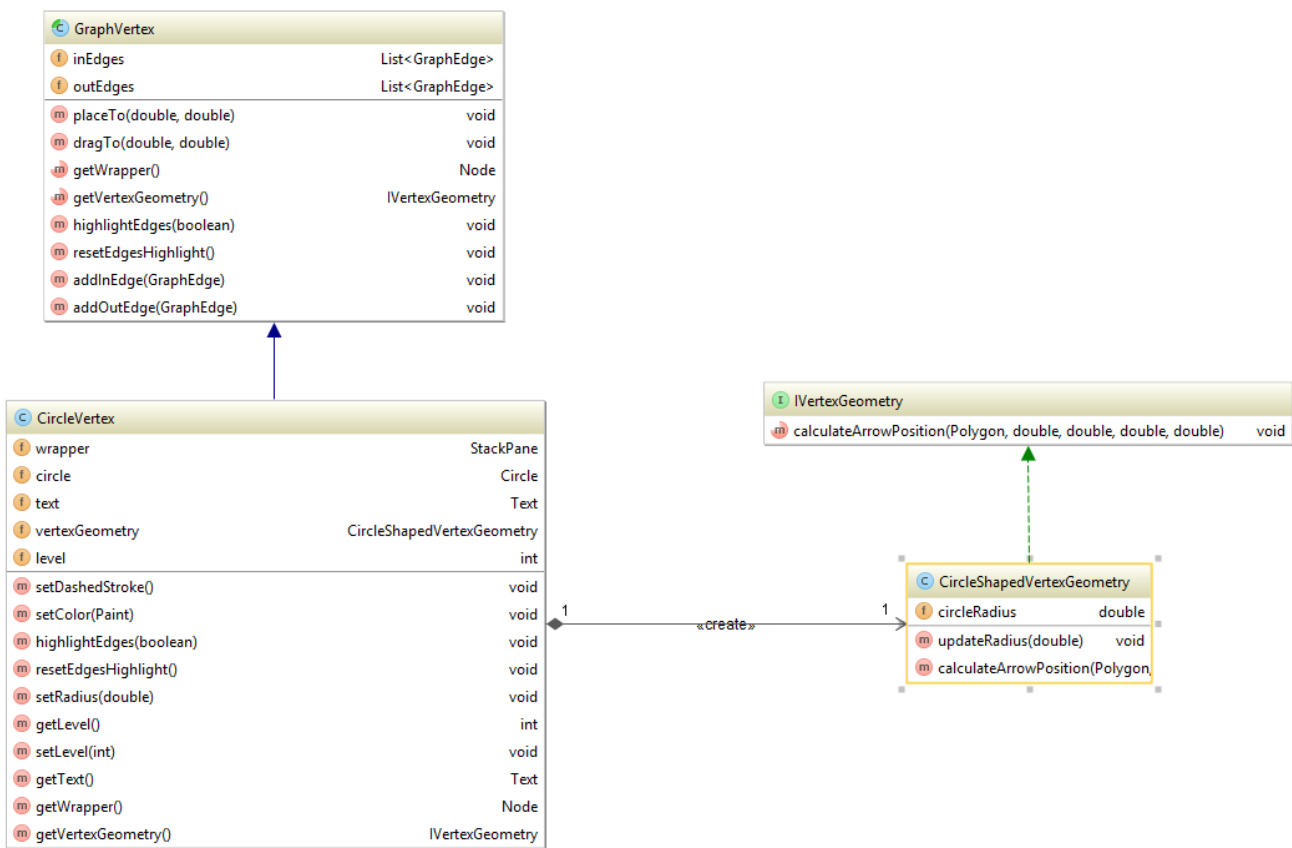


Рисунок 4.30 – Диаграмма классов вершин и метода расчета геометрии

При перерасчете собственных координат, объект ребра графа вызывает метод расчета положения стрелки, определенный для конечной вершины, указатель на которую он содержит. В этом случае объект работает с абстрактными сущностями GraphVertex и IVertexGeometry, в то время как вызовы методов для

них делегируются классам-наследникам. Пример кода вызова метода расчета положения стрелки представлен на рисунке 4.31.

```
/**
 * ...
 * Создание полигона для стрелки
 */

// Расчет координат созданного полигона
endVertex.getVertexGeometry().calculateArrowPosition(arrow, startX, startY, endX, endY);
```

Рисунок 4.31 – Пример кода вызова метода расчета положения стрелки для ребра графа

4.6.3 Обработка событий мыши

Для обработки событий мыши для графа был создан класс `GraphMouseHandler`. Данный класс инкапсулирует три основных события для графа – одиночное нажатие мышью, перетаскивание мышью, отпускание левой кнопки мыши и регистрирует эти события для каждого добавленного элемента графа. Но для того чтобы обеспечить гибкую замену методов обработки сообщений, данный класс делегирует обработку другому объекту, который реализует специальный интерфейс – `IGraphMouseEventStrategy`, содержащий такие методы как `onMouseClicked`, `onMouseDownVertex` и другие. Такой прием является реализацией одного из шаблонов проектирования, под названием «Стратегия». Данный шаблон предполагает реализацию группы взаимозаменяемых алгоритмов с общим интерфейсом, таким образом, что клиенту нет необходимости точно знать какой алгоритм используется – он просто делегирует сообщения через интерфейс. В данном случае, если для `GraphMouseHandler` не определено поле `mouseEventStrategy`, то оно создается как объект класса `DefaultGraphMouseEventStrategy` – стандартная реализация обработки событий от мыши. На рисунке 4.32 представлена диаграмма классов системы обработки событий.

Таким образом, определяя собственную стратегию обработки, реализующую интерфейс `IGraphMouseEventStrategy` можно заменить реакцию на события мыши, определенные по умолчанию.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|------|--------------------|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | Лист |
| | | | | | | | | | | 60 |

фабричного метода Cipher.getInstance, на вход которого передается название алгоритма шифрования (в данном случае это «Blowfish»). У полученного объекта вызывается метод init, в который передается секретный ключ в виде массива байт и флаг режима (ENCRYPT_MODE, DECRYPT_MODE).

Так как при разработке на Java работа с файлами как правило осуществляется с помощью специальных объектов класса Stream (поток), удобнее всего будет управлять шифрованием файлов специальными потоками CipherInputStream и CipherOutputStream.

Для того чтобы выделить операции по шифрованию/дешифрованию файлов создан класс SecurityUtil, имеющий 2 статических поля – defaultKey (секретный ключ по умолчанию, применяемый в случае шифрования файлов без пароля) и algorithm – константа строкового типа со значением «Blowfish». Помимо этого, класс содержит 2 статических метода – encryptFile и decryptFile, которые в качестве параметров принимают секретный ключ (null если пароль не нужен) и поток для файла (input или output), и возвращают CipherOutputStream (позволяет зашифровать переданный на вход поток) и CipherInputStream (позволяет расшифровать переданный на вход поток) соответственно. Диаграмма утилитного класса SecurityUtil представлена на рисунке 4.33.

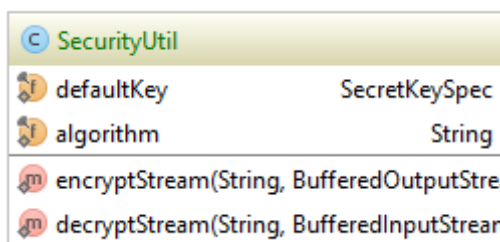


Рисунок 4.33 – Диаграмма утилитного класса SecurityUtil

Выводы

В разделе определены основные сущности и методы, с которыми будет работать программа, и представлены в виде отдельной библиотеки логического вывода. Данная библиотека не имеет зависимостей от текущего проекта и может быть повторно использована для других проектов по данной тематике. В библиотеке представлен механизм многопоточного выполнения процедур логического вывода. При этом не рассматривается эффективность многопоточной реализации. Экспериментально было установлено, что при решении небольших задач логического вывода в логике высказываний многопоточное выполнение процедур не дает ощутимого прироста производительности. Это связано в первую очередь с тем, что процедуры логического вывода в логике высказываний

| | | | | | |
|----------------|--|---|--|--|--|
| Подпись и дата | | | | | |
| Инф. № дубл. | | Рисунок 4.33 – Диаграмма утилитного класса SecurityUtil | | | |
| Взам. инв № | | Выводы | | | |
| Подпись и дата | | <p>В разделе определены основные сущности и методы, с которыми будет работать программа, и представлены в виде отдельной библиотеки логического вывода. Данная библиотека не имеет зависимостей от текущего проекта и может быть повторно использована для других проектов по данной тематике. В библиотеке представлен механизм многопоточного выполнения процедур логического вывода. При этом не рассматривается эффективность многопоточной реализации. Экспериментально было установлено, что при решении небольших задач логического вывода в логике высказываний многопоточное выполнение процедур не дает ощутимого прироста производительности. Это связано в первую очередь с тем, что процедуры логического вывода в логике высказываний</p> | | | |
| Инф. № продл. | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | </ | | | |

выполняются достаточно быстро и в основном машинное время затрачивается на инициализацию потоков и переключение контекста. Однако, стоит отметить, что процедуры логического вывода в логике предикатов выполняются существенно дольше и здесь ключевую роль в вопросе производительности может сыграть их параллельное выполнение.

Разработан декларативный язык логического программирования, позволяющий описать исходную базу знаний в виде правил. Язык имеет достаточно простой синтаксис, который будет привычен пользователям, знакомым с программированием. Правила можно ввести, используя в основном только буквы латинского алфавита, без необходимости частого ввода специальных символов.

Разработана библиотека построения схемы логического вывода, основанная на компонентах технологии JavaFX. При этом, стоит отметить, что библиотека не имеет зависимостей от каких-либо сущностей или методов других модулей разрабатываемой программы. Таким образом она может быть использована в других проектах для построения интерактивных графов, используя при этом в качестве узлов любой графический компонент JavaFX.

Также стоит добавить, что, используя некоторые приемы объектно-ориентированного проектирования можно получить достаточно гибкий программный код, который легко поддерживать и расширять.

| | | | | | | | | | | |
|----------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прораб. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 63 |

базы знаний в виде посылок, а также присутствуют 2 кнопки – загрузка данных из текстового файла и сохранение введенных данных в текстовый файл. Кнопка сохранения становится доступной только в случае, когда поле для ввода содержит какой-либо текст.

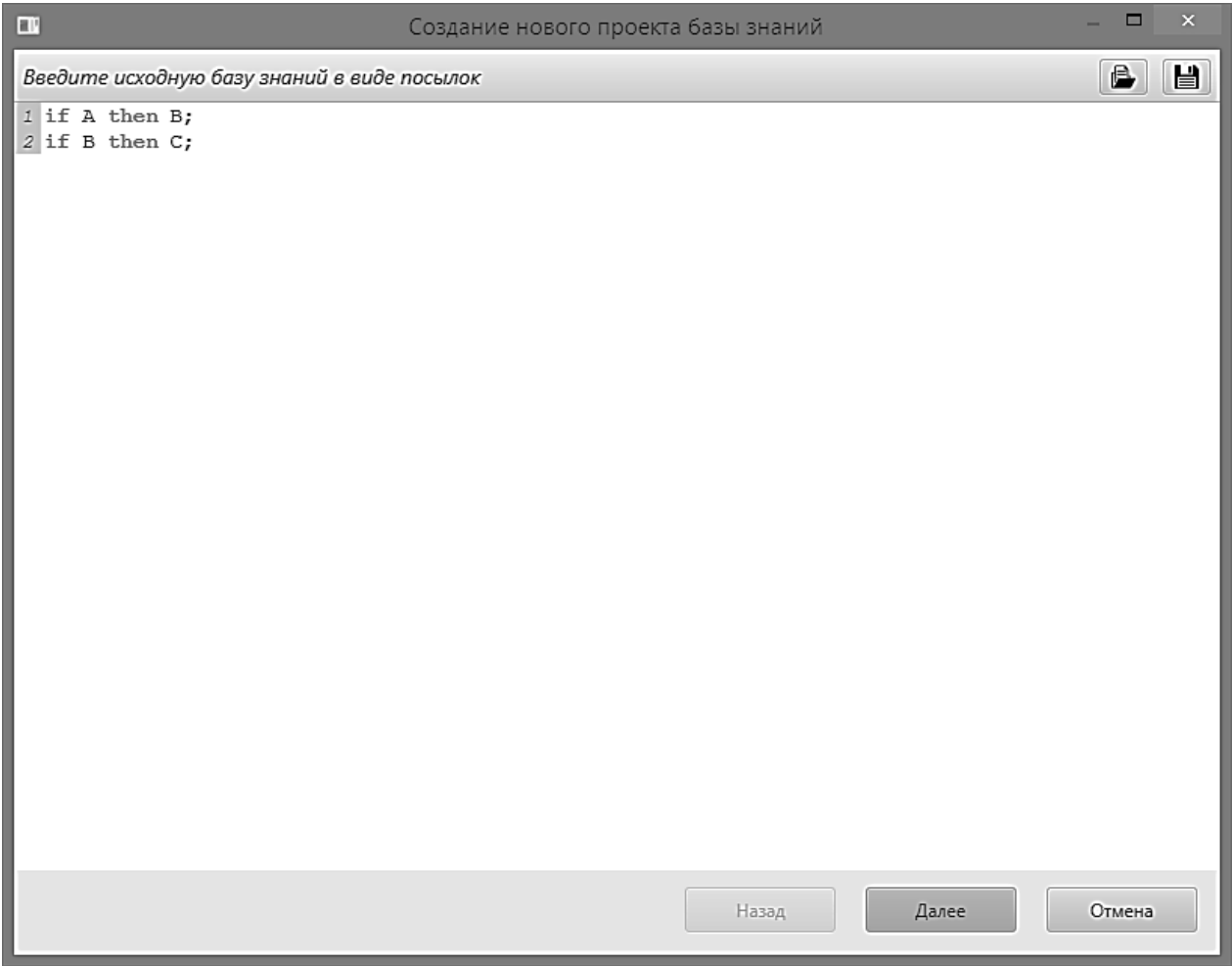
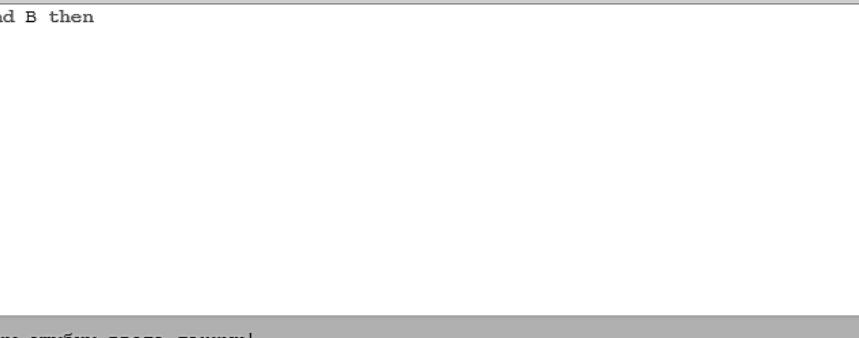


Рисунок 5.2 – Диалоговое окно создания новой базы знаний

Как показано на рисунке 5.3, в случае некорректного ввода исходных данных появляется сообщение об ошибке, которое затем пропадает при следующей установке фокуса на поле для ввода. После того, как исходные правила введены корректно, по нажатию кнопки «Далее» происходит переход на второй шаг создания проекта. На этом шаге каждому литералу из введенных данных можно назначить его текстовую интерпретацию. Для этого необходимо двойным щелчком мыши установить фокус в поле для ввода, которое находится внутри ячейки столбца «Интерпретация», ввести текстовую информацию, затем нажать кнопку «Enter» для завершения ввода. Таблица соответствий литерал-интерпретация представлена на рисунке 5.4. Стоит отметить, что ввод интерпретаций для каждого литерала не обязателен. По нажатию кнопки «Завершить», создается новый проект и диалоговое окно закрывается.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инф. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 65 |
| Изм. | Лист | № докум. | Подпись | Дата | | | | | | |



Создание нового проекта базы знаний

Введите исходную базу знаний в виде посылок

1 if A and B then

Обнаружены ошибки ввода данных!
-- строка 1, элемент 16: некорректная консеквента

Назад Далее Отмена

Рисунок 5.3 – Сообщение об ошибке при вводе исходных данных

Создание нового проекта базы знаний

Каждому литералу можно назначить интерпретацию

| Литерал | Интерпретация |
|---------|---------------|
| A | Человек |
| B | Смертен |
| C | Сократ |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Назад

Завершить

Отмена

Рисунок 5.4 – Таблица интерпретаций литералов

Рабочая область открытого проекта разделена на несколько частей. В верхней части расположена панель, содержащая некоторые элементы управления. В левой части окна расположен список задач. Пример добавления задачи показан на рисунке 5.5.

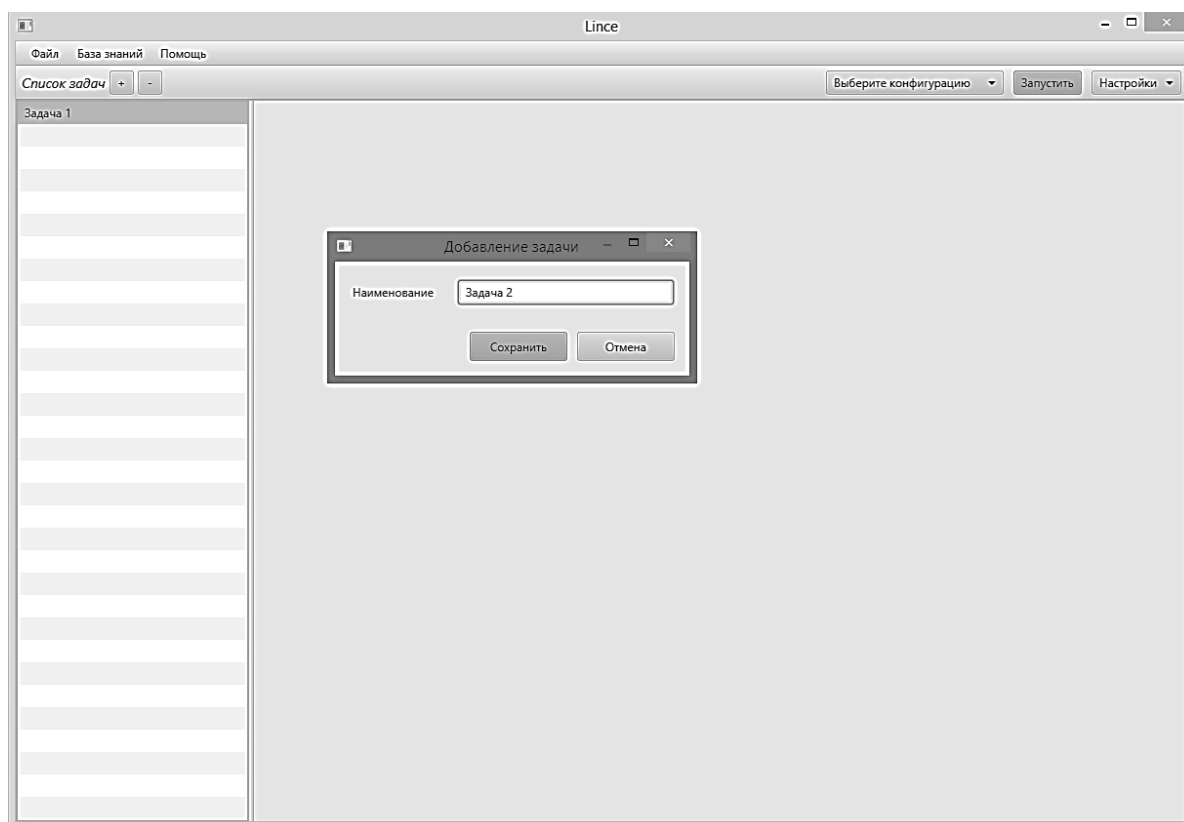


Рисунок 5.5 – Пример добавления задачи

Задачей в данной программе является результат логического вывода для одного заключения. Таким образом, возможна работа с несколькими результатами вывода, которые в свою очередь сохраняются вместе с базой знаний в один файл. При двойном клике мышью на элемент в списке задач в правой части окна разворачивается рабочая область задачи в виде вкладки. Возможно одновременно открыть несколько вкладок как показано на рисунке 5.6. При закрытии вкладки и последующем ее открытии состояние рабочей области остается неизменным. Каждая вкладка содержит верхнюю панель с компонентом для ввода текста – заключения.

Для того, чтобы запустить решение задачи необходимо выбрать конфигурацию с помощью выпадающего списка в панели управления программы. По умолчанию проект создается без конфигураций, поэтому необходимо создать хотя бы одну конфигурацию. Для этого в выпадающем списке «Настройки» панели управления необходимо выбрать пункт «Добавить». При этом откроется диалоговое окно «Добавление конфигурации» как показано на рисунке 5.7.

| | | | | | | | | | | |
|---------------|----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инф. № прокл. | Подпись и дата | Взам. инв. № | Инв. № дубл. | Подпись и дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 67 |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | | | | | | |

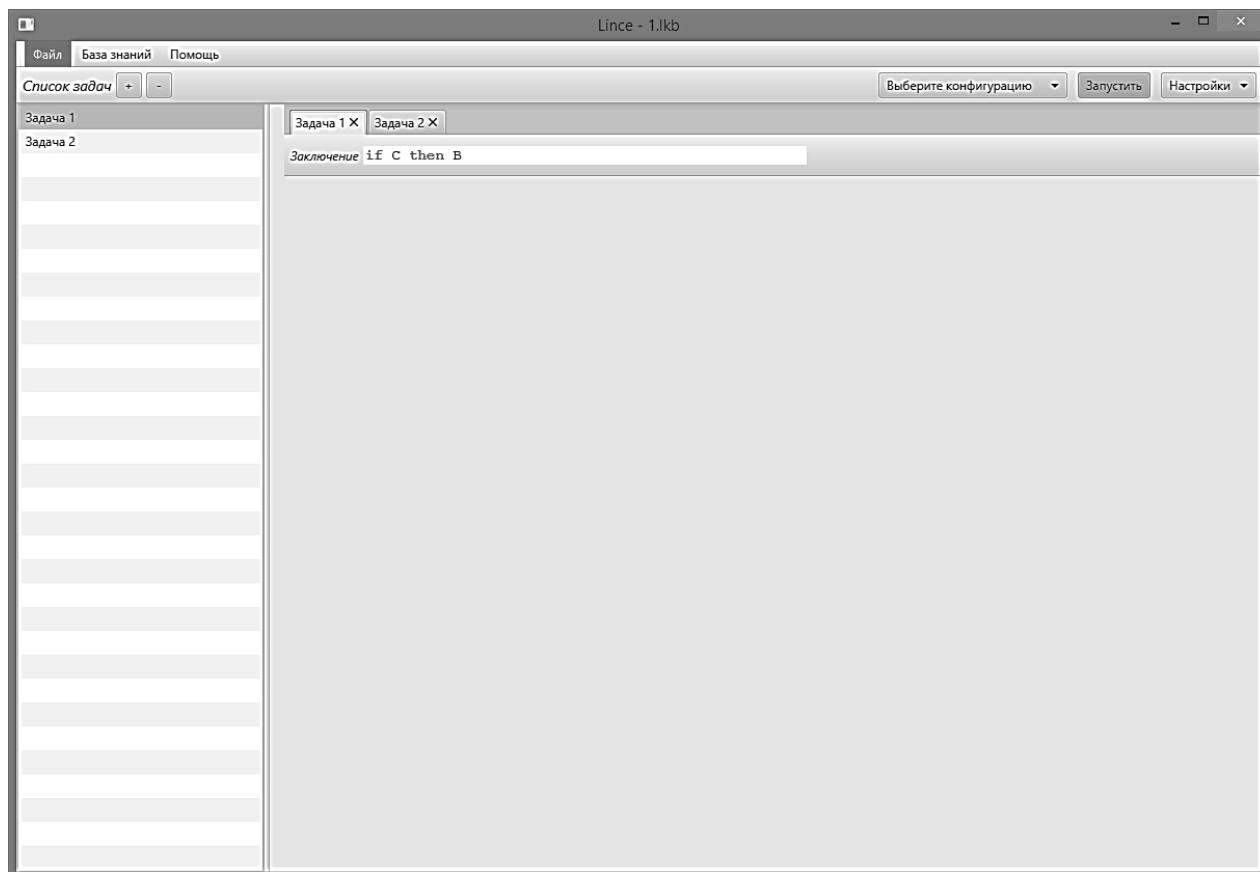


Рисунок 5.6 – Работа с несколькими вкладками задач

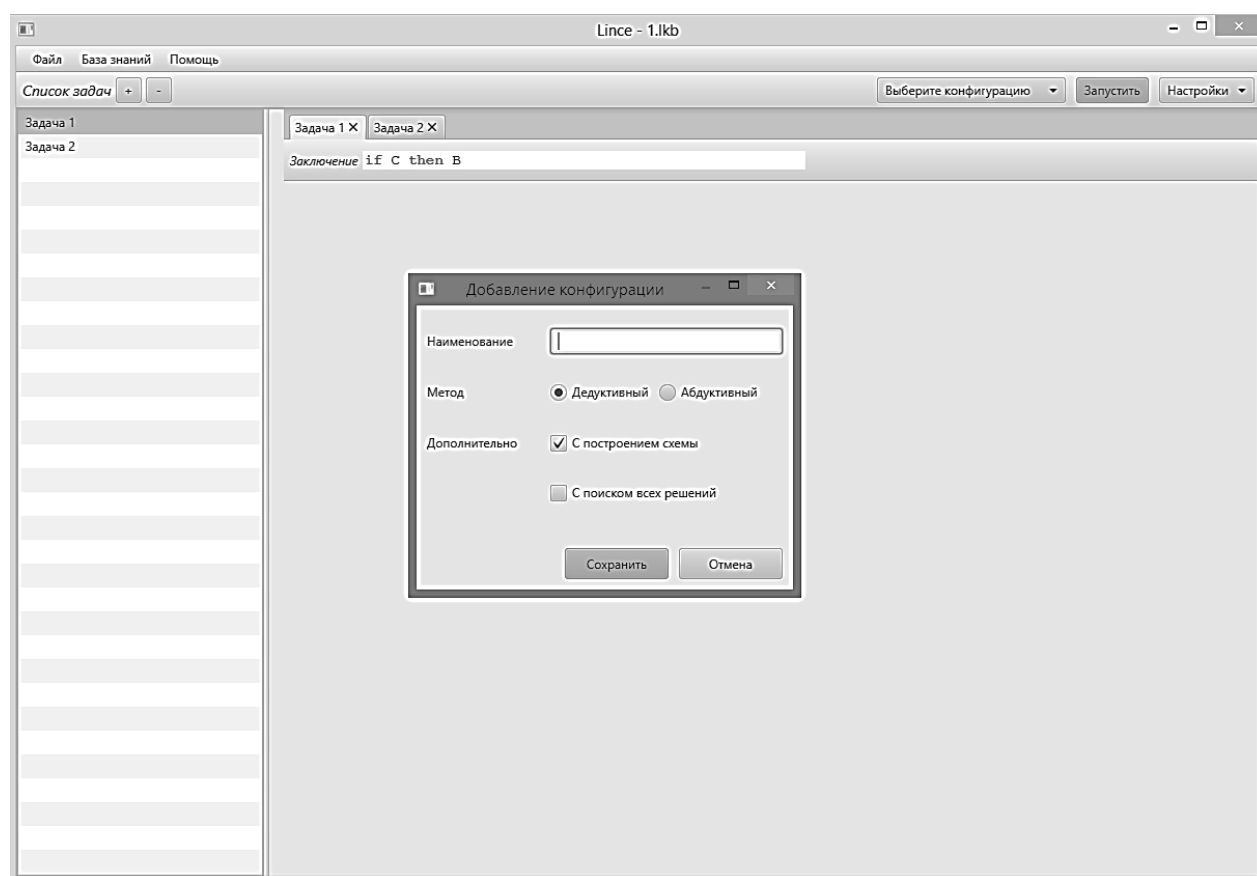


Рисунок 5.7 – Добавление конфигурации

| | |
|----------------|----------------|
| Инв. № продл. | Подпись и дата |
| Взам. инв. № | Инв. № дубл. |
| Подпись и дата | |
| Инв. № продл. | |

| | | | | |
|------|------|----------|---------|------|
| Изм. | Лист | № докум. | Подпись | Дата |
|------|------|----------|---------|------|

При добавлении конфигурации необходимо ввести ее наименование (обязательный параметр), а также выбрать метод и дополнительные параметры. Параметр «С поиском всех решений» доступен только для дедуктивного вывода. После нажатия на кнопку «Сохранить» конфигурация добавляется в выпадающий список и автоматически выделяется в случае если до этого список был пуст. Также присутствует возможность редактирования и удаления конфигураций с помощью выбора соответствующих пунктов в выпадающем списке «Настройки».

По нажатию кнопки «Запустить» выполняется поиск решения для выбранной в данный момент задачи (активная вкладка). В верхней панели рабочей области задачи появляется сообщение о результате решения. Как показано в примере на рисунке 5.8 решение выбранной задачи было найдено, в левой части рабочей области появилось дерево с информацией о ходе вывода, а в правой части была построена схема логического вывода. При этом есть возможность перетаскивания вершин схемы логического вывода. При наведении указателя мыши на символическое представление литерала на схеме появляется его интерпретация в виде подсказки. Вдобавок, есть возможность те или иные дуги графа, которые были сформированы при выполнении какой-либо из процедур вывода. Для этого необходимо выделить нужную процедуру в дереве вывода, щелкнув по ней левой кнопкой мыши.

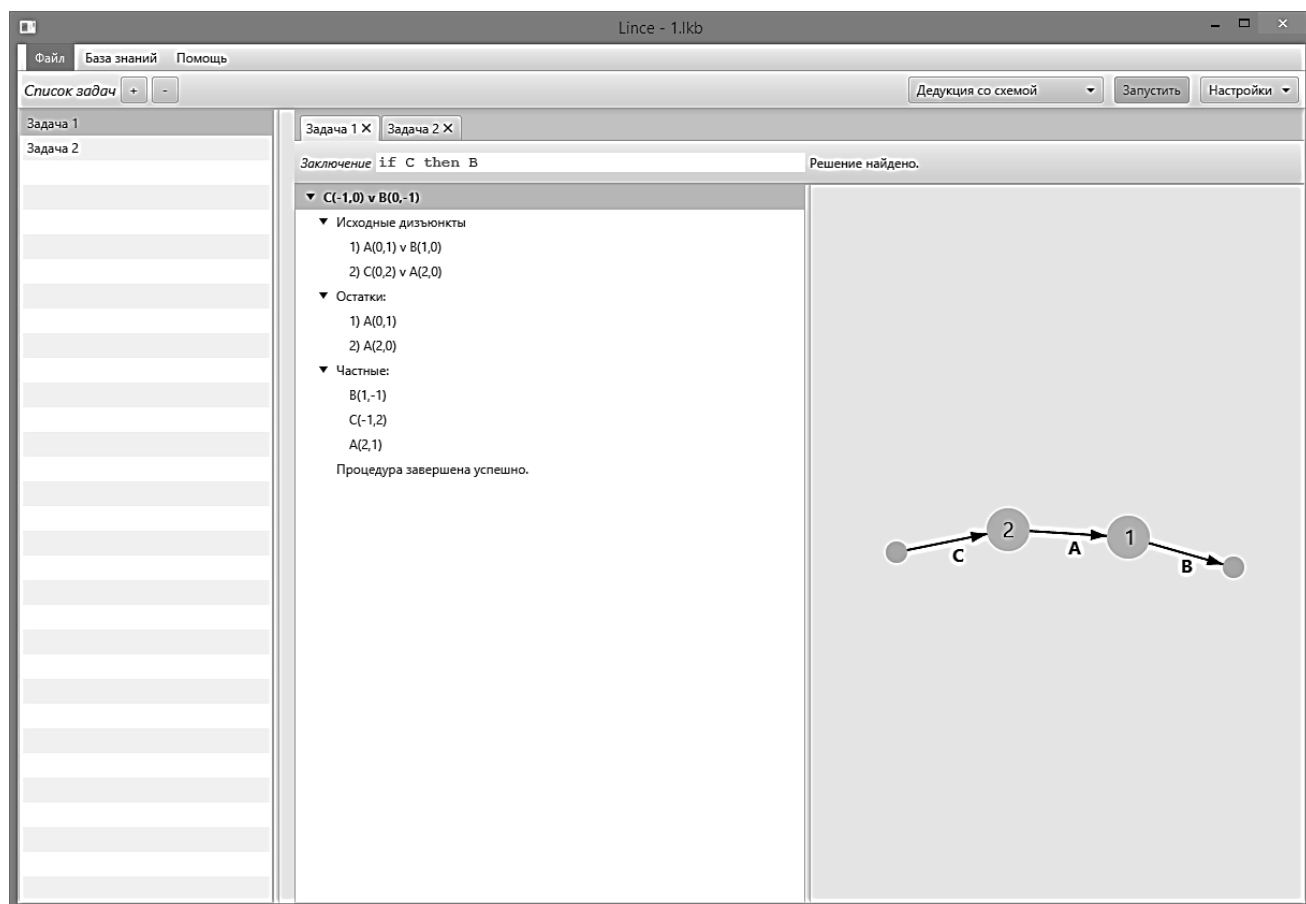


Рисунок 5.8 – Пример решения задачи

Подпись и дата

Инв. № дубл.

Взам. инв. №

Подпись и дата

Инв. № прошл.

| | | | | |
|------|------|----------|---------|------|
| | | | | |
| Изм. | Лист | № докум. | Подпись | Дата |

ТПЖА.230101.016 ПЗ

Лист

69

Пункт меню «База знаний» содержит такие опции как «Редактировать базу знаний», «Интерпретация литералов», «Создать/Изменить пароль». При выборе опции «Редактировать базу знаний» открывается модальное окно, представленное на рисунке 5.9.

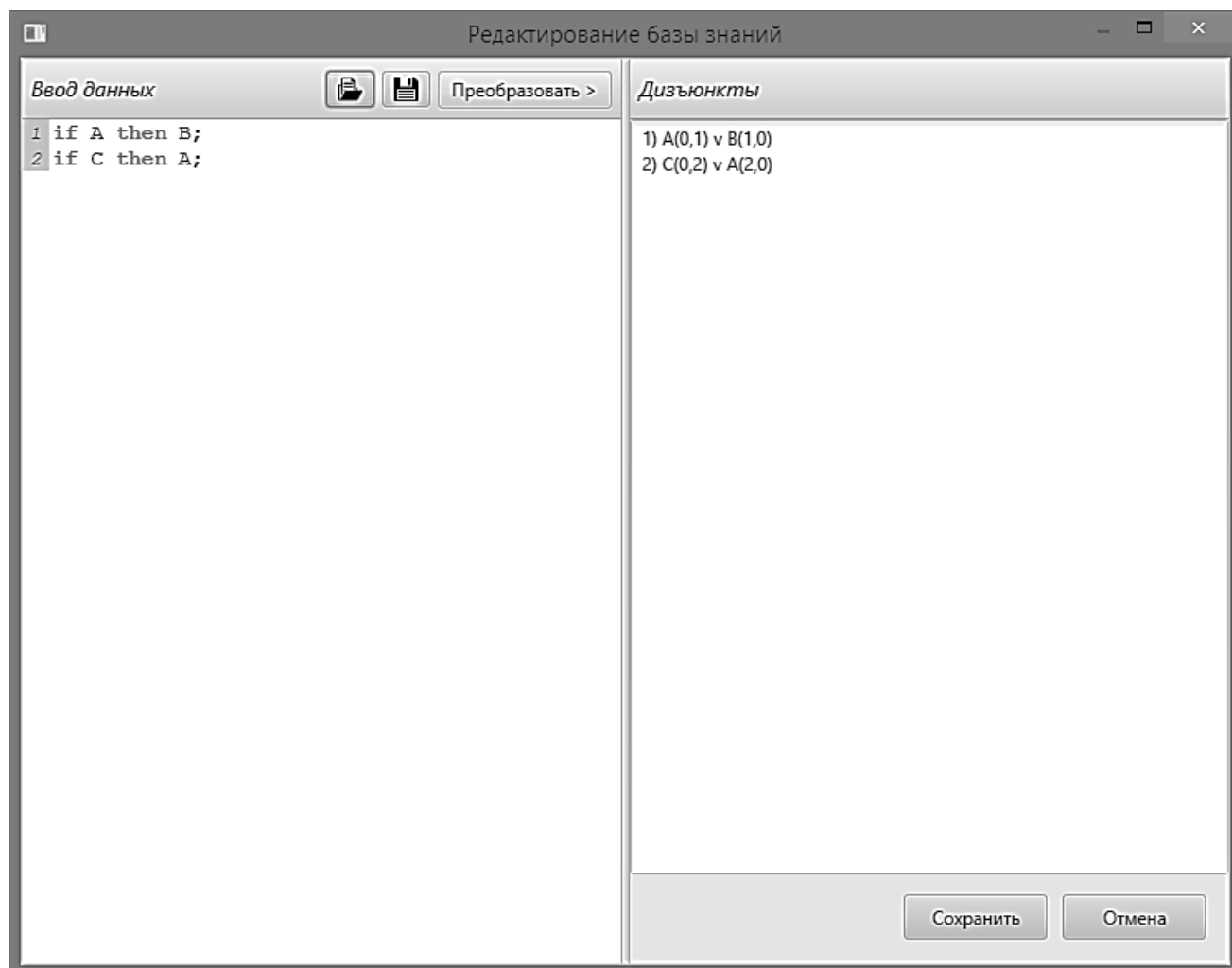


Рисунок 5.9 – Окно редактирования базы знаний

С помощью формы, представленной на рисунке 5.9, можно добавлять, редактировать или удалять исходные правила. После ввода правил с помощью компонента для ввода текста в левой части окна необходимо нажать на кнопку «Преобразовать» для того, чтобы осуществить проверку правильности введенных данных и преобразовать их в форму дизъюнктов. После этого, по нажатию кнопки «Сохранить» база знаний модифицируются и новые задачи могут запускаться на выполнение, используя новую базу знаний.

При выборе опции «Интерпретации литералов» открывается модальное окно, которое содержит форму редактирования, аналогичную представленной на рисунке 5.4.

| | | | | | | | | | | |
|---------------|----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата | Взам. инв. № | Инв. № дубл. | Подпись и дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 70 |
| Изм. | Лист | № докум. | Подпись | Дата | | | | | | |

| | | | | |
|---------------|-----------------|-------------|--------------|----------------|
| Инв. № прокл. | Подпись и дата. | Взам. инв № | Инв. № дубл. | Подпись и дата |
| | | | | |



5.2 Результаты экспериментальной апробации

ТПЖА.230101.016 ПЗ

представлено на рисунке 5.11. Формирование интерпретации литералов представлено на рисунке 5.12. Результат решения задачи с помощью абдуктивного логического вывода показан на рисунке 5.13.

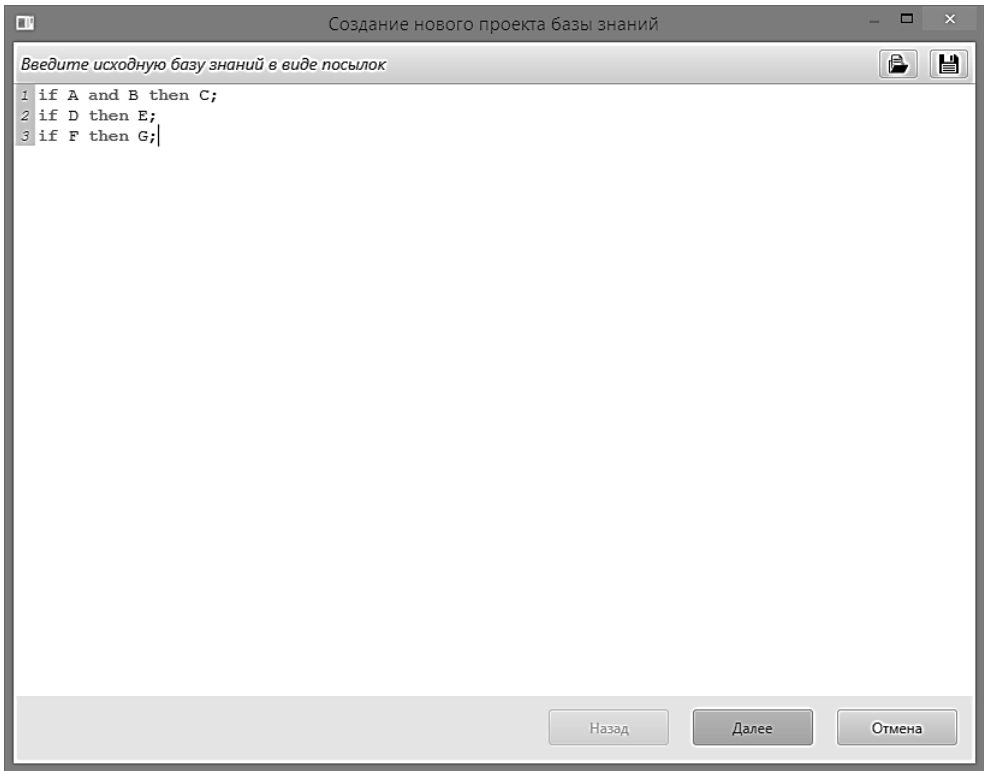


Рисунок 5.11 – Описание задачи в символьном виде

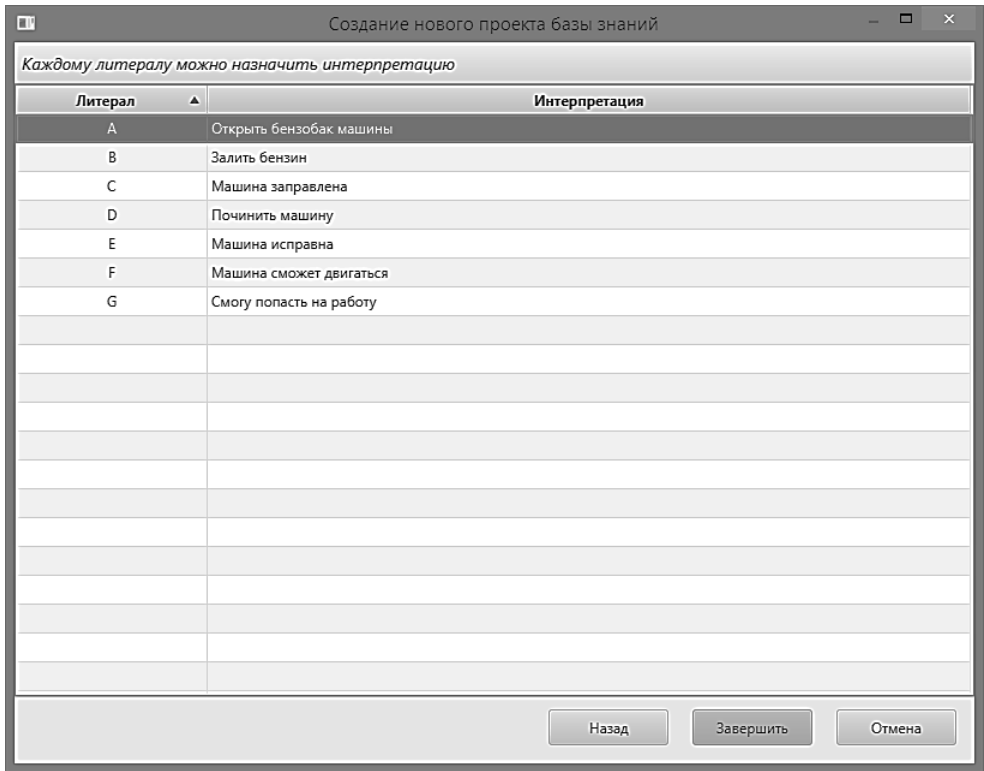


Рисунок 5.12 – Формирование интерпретации литералов

| | |
|----------------|----------------|
| Инв. № прокл. | Подпись и дата |
| Взам. инв. № | Инв. № дубл. |
| Подпись и дата | |

| | | | | |
|------|------|----------|---------|------|
| Изм. | Лист | № докум. | Подпись | Дата |
|------|------|----------|---------|------|

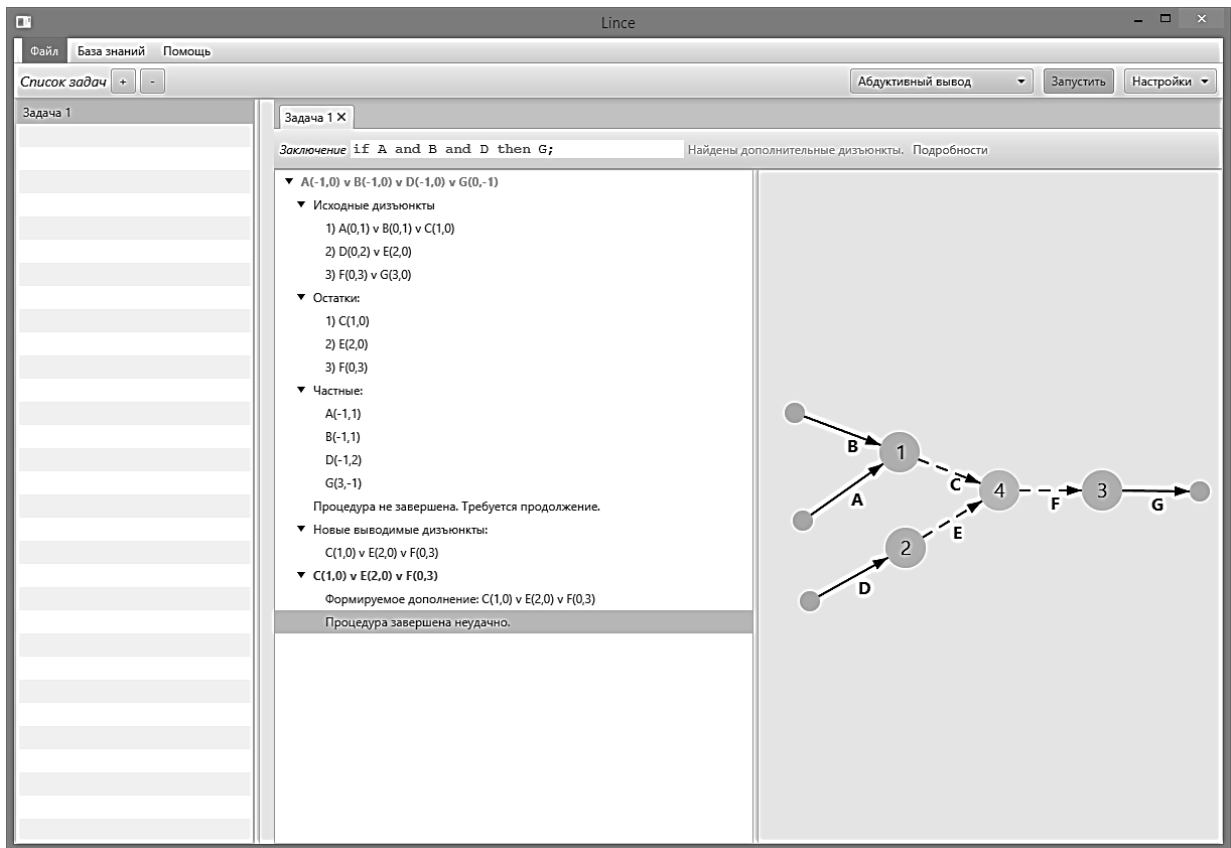


Рисунок 5.13 – Результат решения задачи

Как показано на рисунке 5.13 абдуктивный логический вывод завершился неудачно, однако появилось сообщение о том, что найдены дополнительные дизъюнкты. Также, на схеме вывода была образована новая вершина, представляющая собой дополнительное правило, сформированное в ходе абдуктивного логического вывода. По нажатию на ссылку «Подробности» рядом с сообщением открывается диалоговое окно, представленное на рисунке 5.14.

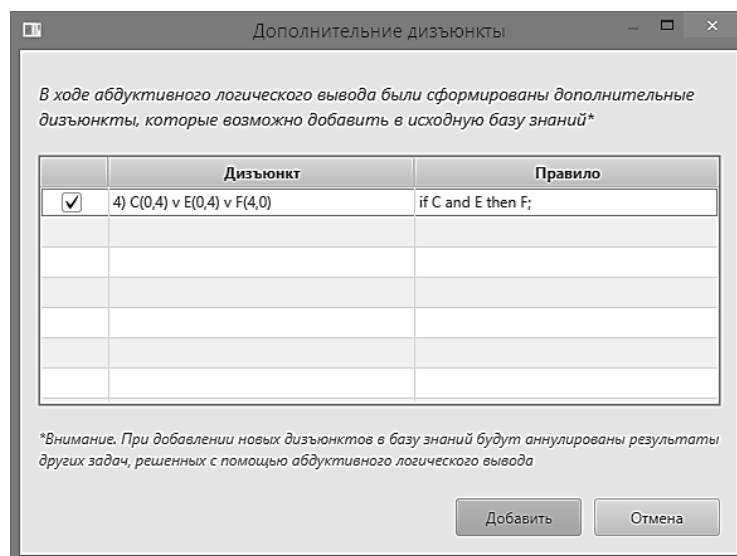


Рисунок 5.14 – Диалоговое окно добавления дополнительных правил

После добавления дополнительного правила «Если машина будет заправлена и исправна, то машина сможет двигаться.» в базу знаний можно повторно запустить решение задачи. На этот раз абдуктивный вывод завершается успешно, как показано на рисунке 5.15.

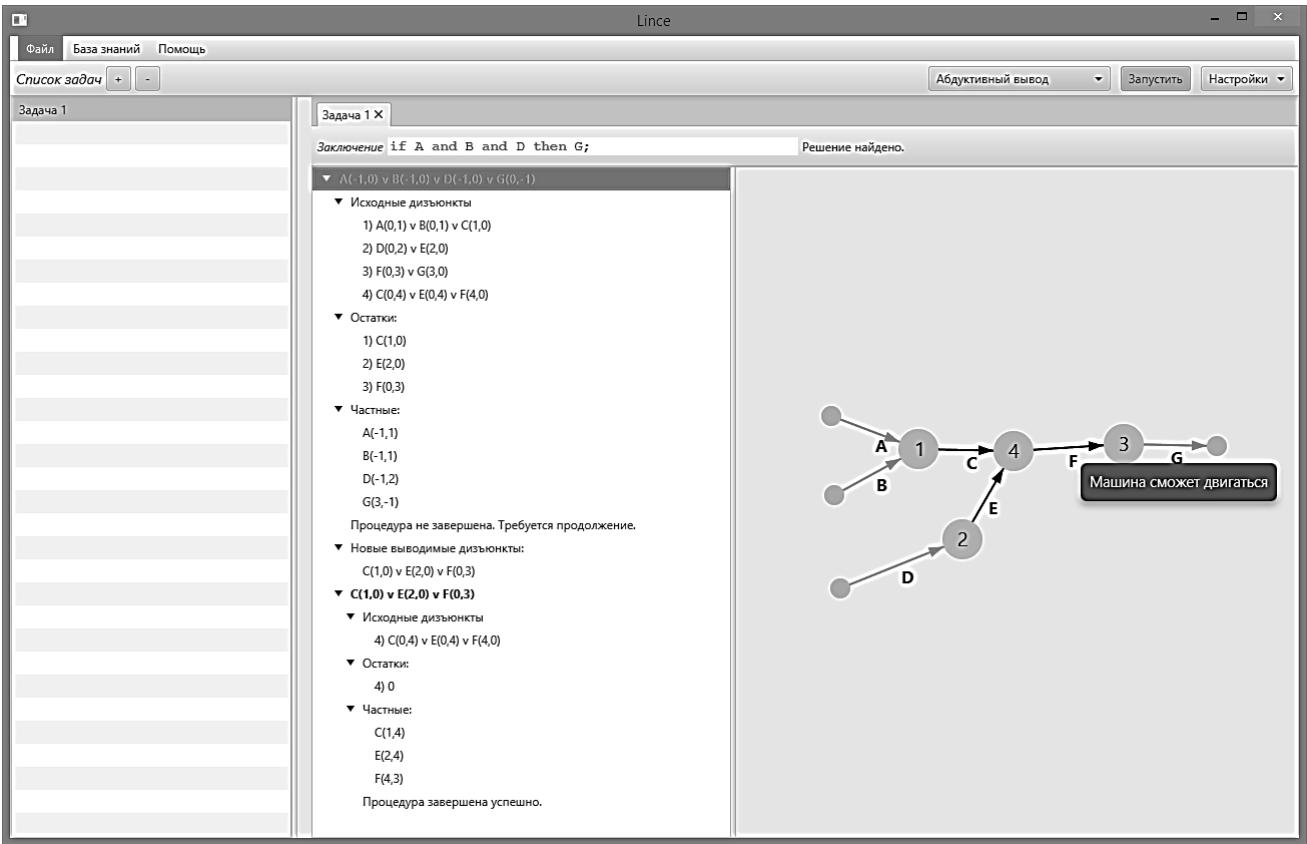


Рисунок 5.15 – Результат решения задачи

Также, как видно из рисунка 5.15 графическое представление схемы позволяет отобразить интерпретацию конкретного литерала в виде подсказки, которая появляется при наведении указателя мыши на символическое представление литерала.

В рамках апробации дедуктивного логического вывода для решения была выбрана задача, записанная в символическом виде в монографии[1]. Описание задачи представляется в виде 14 простых правил, которые представлены на рисунке 5.16. Результат решения задачи с помощью дедуктивного логического вывода с построением схемы представлен на рисунке 5.17. Как видно из рисунка 5.17 дедуктивный вывод для задачи завершился успешно. В ходе вывода сформирована схема, состоящая из 14 основных вершин, отражающих исходные правила и 5 вершин меньшего размера, которые отражают составляющие правила заключения.

| | |
|----------------|----------------|
| Инв. № дубл. | Подпись и дата |
| Взам. инв. № | |
| Инв. № подл. | |
| Подпись и дата | |
| Инв. № подл. | |

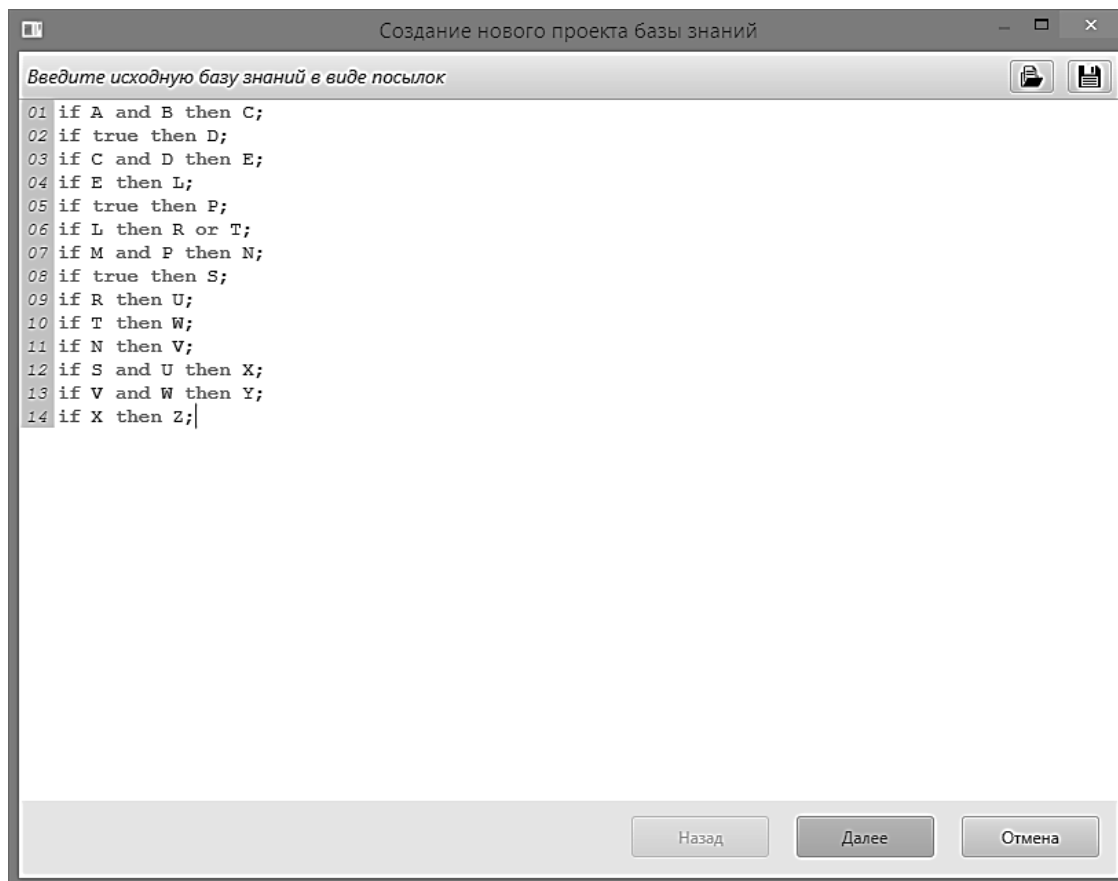


Рисунок 5.16 – Описание правил задачи

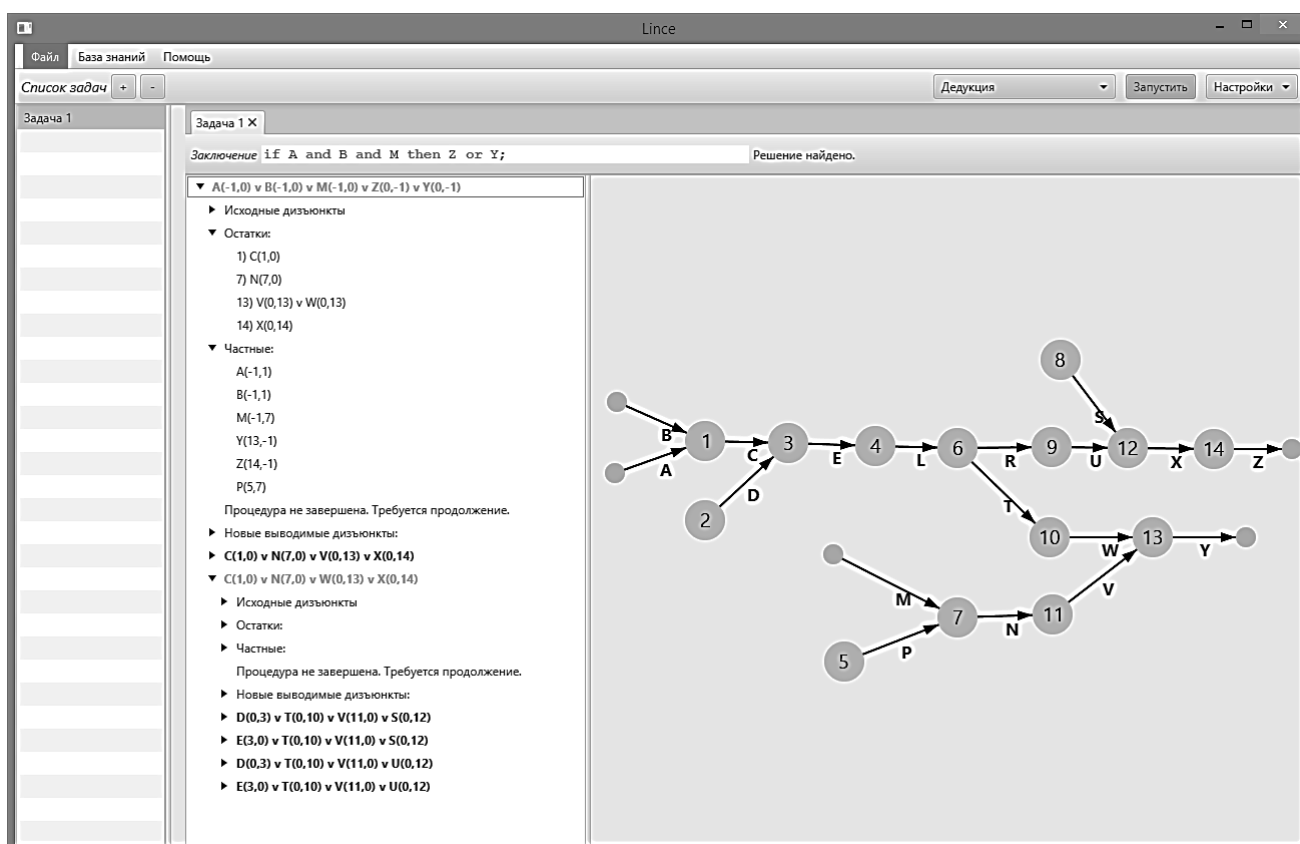


Рисунок 5.17 – Результат решения задачи

5.3 Минимальные системные требования для программы

Минимальные системные требования, необходимые для запуска и приемлемой работы программы основаны на рекомендуемых системных требованиях для платформы Java 8:

- центральный процессор Intel Pentium 4 2.4 ГГц или аналогичный AMD Athlon 64 +2800;
- объем оперативной памяти: 1 Гб (рекомендуется 2 Гб);
- 300 Мб свободного дискового пространства;
- дисплей с разрешением 800x600 точек и выше;
- операционная система Windows Vista/7/8/8.1 или Mac OS X 10.8.3 и выше или Ubuntu 12.04 LTS, Linux Mint 17 и выше;
- клавиатура, мышь.

Для использования всех преимуществ JavaFX в рендере графических компонентов система должна быть оснащена одним из графических ускорителей, представленных в таблице 5.1

Таблица 5.1 – Поддерживаемые графические ускорители для JavaFX

| Вендор | Модели, версии |
|--------|---|
| NVIDIA | Серии GeForce 8 или 100 и выше, GeForce 8M или 100M и выше, NVS 2100M и выше, Mobility Quadro FX 300M и выше. |
| ATI | Radeon HD 2400, 3000, 4000, 5000, 6000 серии и выше |
| Intel | Серии GMA 4500, GMA 4500MHD, GMA HD и выше |

Выводы

Стоит заметить, что программа имеет достаточно удобный интерфейс для исследования методов логического вывода делением дизъюнктов с построением схемы. Отдельные задачи разделяются по вкладкам, что позволяет работать с несколькими заключениями в рамках одной исходной базы правил. Интерактивные возможности построенной схемы логического вывода позволяют расположить ее элементы в любом порядке. При вводе правил с помощью разработанного декларативного языка осуществляется подсветка ключевых слов синтаксиса.

Проанализировав минимальные системные требования, можно сделать вывод о том, что программа будет запускаться и работать с приемлемой скоростью на всех современных компьютерах, а также на большинстве компьютеров, выпущенных 5-7 лет назад.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 76 |

6 Финансово-экономическое обоснование необходимости разработки ПО

В данном разделе представлен состав команды разработки проекта с указанием времени участия каждого члена команды, произведен подробный расчет затрат на создание программного обеспечения, а также составлен график выполнения работ по проекту.

6.1 Расчет затрат на создание программного обеспечения

При расчете затрат предполагается, что разработка ведется коммерческой или бюджетной организацией, состоящей из должностных лиц, представленных в таблице 6.1.

Данные по месячному окладу соответствуют средней статистике по г. Киров.

Таблица 6.1 – Состав команды разработки ПО

| Наименование должности | Численность, чел. | Повышающий коэффициент | Месячный оклад, руб |
|-----------------------------|-------------------|------------------------|---------------------|
| Руководитель | 1 | 1,8 | 15 000 |
| Ведущий инженер-программист | 1 | 1,7 | 12 000 |

Время участия каждого специалиста в создании ПО определяется перечнем работ и трудоемкостью их выполнения. Трудоемкость выполнения работ определяется на основе экспертных оценок по формуле

$$t_p = \frac{3 \cdot t_{min} + 2 \cdot t_{max}}{5}, \quad (6.1)$$

где t_p – расчетная трудоемкость выполнения работы;

t_{min} – минимальное время, необходимое для выполнения работы;

t_{max} – максимальное время, необходимое для выполнения работы.

Результаты расчетов трудоемкости по созданию программного обеспечения приведены в таблице 6.2.

| | | | | | |
|----------------|-----------------|--------------|--------------|----------------|--------------------|
| Инв. № прораб. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | |
| | | | | | |
| | | | | | |
| | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ |
| | | | | | Лист 77 |

Таблица 6.2 – Трудоемкость выполнения работ по созданию ПО

| Наименование работы | t _{min} , час | t _{max} , час | t _p , час | В том числе | |
|---|---------------------------|---------------------------|-------------------------|--------------|---------------------|
| | | | | руководитель | ведущий программист |
| Разработка требований к программному продукту, составление технического задания | 18 | 26 | 22 | 14 | 8 |
| Изучение литературы и научных работ по предметной области | 20 | 33 | 26 | 4 | 22 |
| Выбор и изучение инструментальных средств разработки | 16 | 28 | 21 | 0 | 21 |
| Разработка алгоритмов функционирования программного продукта | 24 | 48 | 34 | 5 | 29 |
| Разработка макетов интерфейса пользователя | 20 | 30 | 24 | 0 | 24 |
| Написание программы | 250 | 400 | 310 | 0 | 310 |
| Отладка и тестирование программы | 100 | 150 | 120 | 0 | 120 |
| Разработка документации и чертежей | 24 | 48 | 34 | 14 | 20 |
| Разработка инструкции пользователя | 15 | 30 | 21 | 0 | 21 |
| Финансово-экономическое обоснование разработки | 8 | 16 | 12 | 12 | 0 |
| Оформление документации с использованием ЭВМ | 16 | 24 | 20 | 5 | 15 |
| Маркетинговые исследования | 40 | 64 | 50 | 40 | 10 |
| Всего | 551 | 897 | 694 | 94 | 600 |

Для удобства и точности проведения дальнейших расчетов работы, приведенные данные в таблице 6.2, целесообразно сгруппировать по комплексам и записать их по форме, указанной в таблице 6.3.

| | | | | | | | | | | |
|----------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прораб. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 78 |
| Изм. | Лист | № докум. | Подпись | Дата | | | | | | |

Таблица 6.3 – Комплексы работ по созданию ПО

| Наименование комплекса работ | Обозначение | t _р , час | В том числе | |
|---|-----------------|----------------------|--------------|---------------------|
| | | | руководитель | ведущий программист |
| Создание структуры ПО, математического обеспечения и написание алгоритмов | В _{МО} | 127 | 23 | 104 |
| Ввод программы в ЭВМ, редактирование, трансляция, отладка, тестирование, выполнение | В _М | 430 | 0 | 430 |
| Прочие затраты по разработке ПО | В _{ПР} | 12 | 12 | 0 |
| Оформление программного продукта | В _{ОФ} | 75 | 19 | 56 |
| Маркетинговые исследования | В _{МИ} | 50 | 40 | 10 |
| Всего | В _{ПО} | 694 | 94 | 600 |

Общие затраты на создание ПО определяются по формуле

$$З_{СП} = З_{РП} + З_{ОФ} + З_{МИ} , \quad (6.2)$$

где $З_{СП}$ – общие затраты на создание ПО;

$З_{РП}$ – затраты на разработку ПО;

$З_{ОФ}$ – затраты на оформление программного продукта и подготовку его к продаже. Их можно принять в размере 15-25% от $З_{РП}$.

$З_{МИ}$ – затраты на маркетинговые исследования, их можно принять в размере 10-20% от $З_{РП}$.

Общий фонд оплаты труда работников, участвующих в создании ПО, определяется по формуле

$$\Phi OT_{ОБ} = \sum_{i=1}^n \frac{В_{ПО} \cdot F \cdot O_M}{D_S \cdot D_P} + П + В_{РК} , \quad (6.3)$$

где n – количество специалистов;

$В_{ПО}$ – время участия специалиста определенной квалификации в создании ПО, в час;

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | <p>Общие затраты на создание ПО определяются по формуле</p> $З_{СП} = З_{РП} + З_{ОФ} + З_{МИ} , \quad (6.2)$ <p>где $З_{СП}$ – общие затраты на создание ПО; $З_{РП}$ – затраты на разработку ПО; $З_{ОФ}$ – затраты на оформление программного продукта и подготовку его к продаже. Их можно принять в размере 15-25% от $З_{РП}$. $З_{МИ}$ – затраты на маркетинговые исследования, их можно принять в размере 10-20% от $З_{РП}$. Общий фонд оплаты труда работников, участвующих в создании ПО, определяется по формуле</p> $\Phi OT_{ОБ} = \sum_{i=1}^n \frac{В_{ПО} \cdot F \cdot O_M}{D_S \cdot D_P} + П + В_{РК} , \quad (6.3)$ <p>где n – количество специалистов; $В_{ПО}$ – время участия специалиста определенной квалификации в создании ПО, в час;</p> | | | | | Лист |
| | | | | | <p>ТПЖА.230101.016 ПЗ</p> | | | | | 79 |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | | | | | | |

Ф – число специалистов определенной квалификации, участвующих в создании ПО;

ОМ – месячный оклад работника в соответствии с его категорией;

DS – длительность смены (8 часов);

ДР – среднее число рабочих дней в месяце, принимается 21 день;

П – премия, предусмотренная для работников, участвующих в создании программного продукта, принимается в размере 25% от ЗПОБ;

ВРК – выплаты по районному коэффициенту (установлены для г. Кирова в размере 15 % от суммы ЗПОБ и П).

Затраты на разработку ПО рассчитываются по формуле

$$З_{РП} = З_{МО} + З_{КОМ} + З_{ПР} , \quad (6.4)$$

где $З_{МО}$ – затраты на создание математического обеспечения и написание программы;

$З_{КОМ}$ – затраты, связанные с работой компьютера при разработке ПО (ввод программы в ЭВМ, ее трансляция и редактирование, отладка, тестирование, корректировка и выполнение ПО);

$З_{ПР}$ – прочие затраты, связанные с разработкой ПО (изучение задания, литературы, патентов, анализ проблемы и существующих алгоритмов, проведение экономических расчетов и др.), их решено принять в размере 25 % от $З_{МО}$.

Затраты на создание математического обеспечения и написание программы определяются по формуле

$$З_{МО} = ЗП_{МО} + П + В_{РК} + О_{СН} \cdot ФОТ_{МО} + Н_{Р} , \quad (6.5)$$

где $ЗП_{МО}$ – затраты на выплату заработной платы работникам, участвующим в создании математического обеспечения и написании программы;

$О_{СН}$ – ставка страховых взносов, равная 30%. С учетом фактора риска увеличиваем на 0,2 %;

$ФОТ_{МО}$ – фонд оплаты труда работников, участвующих в создании математического обеспечения и написании программы;

$Н_{Р}$ – накладные расходы организации, где разрабатывается ПО, принимаются в размере 100 % от $ЗП_{МО}$.

Затраты на заработную плату работников, участвующих в создании математического обеспечения и написании программы, определяются по формуле

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|-----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата. | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 80 |

$$ЗП_{МО} = ЗП_{ПР} + ЗП_{ДР}, \quad (6.6)$$

где $ЗП_{ПР}$ – заработная плата программистов, участвующих в создании математического обеспечения и написании программы;

$ЗП_{ДР}$ – заработная плата других работников временной творческой группы, принимающих участие в разработке математического обеспечения и написании программы (руководитель группы).

Затраты на зарплату программистов рассчитываются по формуле

$$ЗП_{ДР} = (N_{ОП} \cdot t_c \cdot K_H + t_{ДР}) \cdot \frac{O_M}{D_S \cdot D_P}, \quad (6.7)$$

где $N_{ОП}$ – количество операторов (команд) в программе;

t_c – средняя (стандартная) трудоемкость разработки одного оператора (команды) для используемого языка программирования;

K_H – коэффициент новизны разрабатываемой программы;

$t_{ДР}$ – трудоемкость других видов работ, выполняемых программистами при создании математического обеспечения и написании программы (разработки общих принципов построения программы, ее структуры, входных и выходных форм и др.).

Затраты на выплату зарплаты других работников временной творческой группы, участвующих в разработке математического обеспечения и написании программы, определяются по формуле

$$ЗП_{ДР} = P \cdot B_{МО} \cdot \frac{O_M}{D_S \cdot D_P}, \quad (6.8)$$

где P – число работников определенной квалификации, участвующих в разработке математического обеспечения и написании программы;

$B_{МО}$ – время участия работника определенной квалификации в разработке математического обеспечения и написании программы в часах.

Затраты, связанные с работой компьютера при разработке ПО, можно рассчитать укрупненно по формуле

$$З_{КОМ} = З_E + З_A, \quad (6.9)$$

где $З_E$ – затраты на электроэнергию, потребляемую компьютерами;

$З_A$ – амортизационные отчисления от стоимости парка компьютеров.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|-----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата. | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 81 |

Затраты на электроэнергию, потребляемую компьютерами, определяются по формуле

$$З_E = N \cdot В_{ПО} \cdot C, \quad (6.10)$$

где N – установленная мощность одного компьютера, кВт (0,45 кВт);

C – стоимость одного кВт/час, руб. (5,65 руб.).

Затраты на амортизационные отчисления определяются по формуле

$$З_A = T_{ПР} \cdot n \cdot \frac{S}{T_{АМ}}, \quad (6.11)$$

где T_{ПР} – продолжительность осуществления проекта, месяц;

n – число компьютеров, используемых при разработке ПО, штук (два);

S – стоимость одного компьютера, руб. (20000 руб.);

T_{АМ} – срок полезного использования компьютеров, месяцев (36).

Для расчета зарплаты программистов необходимо воспользоваться формулой (6.8), поскольку определить количество операторов в программе не представляется возможным. Таким образом производится расчет зарплаты участников проекта

$$З_{ПРУК} = 1 \cdot 94 \cdot \frac{15000}{8 \cdot 21} = 8392,92 \text{ руб.} \quad (6.12)$$

$$З_{ПРУК} = 1 \cdot 600 \cdot \frac{12000}{8 \cdot 21} = 42857,14 \text{ руб.} \quad (6.13)$$

Далее вычисляются затраты, связанные с работой двух компьютеров при разработке ПО по формуле (6.10)

$$З_E = 2 \cdot 0,45 \cdot 694 \cdot 5,65 = 3528,99 \text{ руб.} \quad (6.14)$$

Для наглядности расчет затрат сведен в таблицу 6.4.

Таблица 6.4 – Смета затрат на создание ПО

| Наименование статьи затрат | Обозначение | Сумма, руб |
|----------------------------|-------------------|------------|
| Зарплата руководителя | З _{ПРУК} | 8392,85 |

| | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | |
| | | | | | | | | | |
| | | | | | ТПЖА.230101.016 ПЗ | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | 82 | | | | |

Продолжение таблицы 6.4

| Наименование статьи затрат | Обозначение | Сумма, руб |
|---|-------------------|------------|
| Зарплата ведущего программиста | ЗП _{ВПП} | 42857,14 |
| Итого | – | 51249,99 |
| Премия | П | 10250 |
| Выплаты по районному коэффициенту | В _{РК} | 9225 |
| Страховые взносы | О _{СН} | 21358,95 |
| Итого затрат на зарплату сотрудникам | З _{МО} | 92083,94 |
| Затраты, связанные с работой компьютера при разработке ПО | З _{КОМ} | 3529,99 |
| Итого затрат на разработку ПО | З _{РП} | 95613,93 |
| Прочие производственные расходы | П _{ПР} | 42857,14 |
| Всего затрат на создание ПО | З _{СП} | 138471,07 |

На рисунке 6.1 приведена круговая диаграмма, отображающая структуру себестоимости программного продукта.

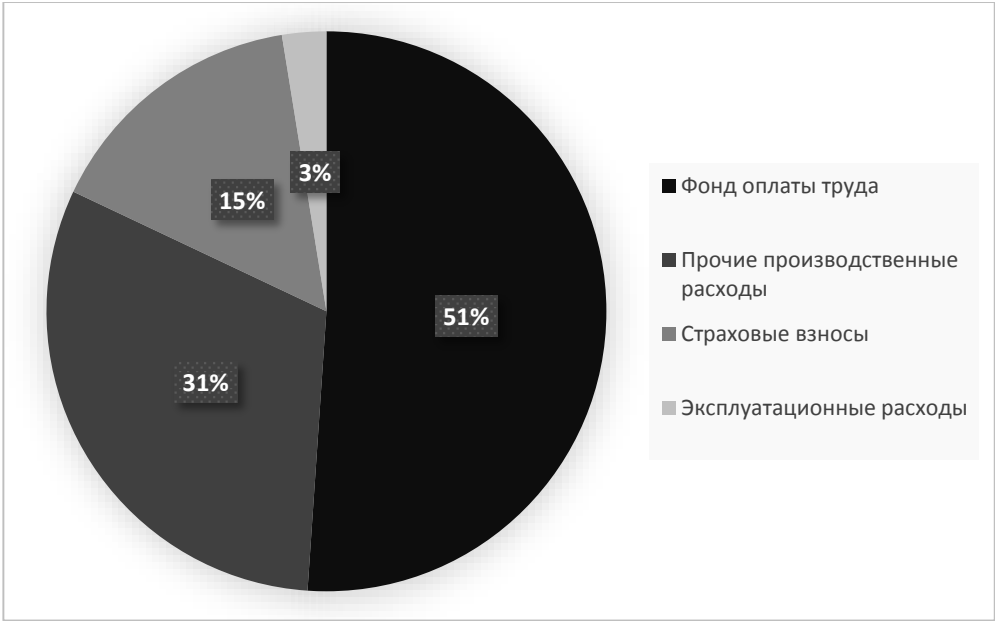


Рисунок 6.1 – Структура себестоимости программного продукта

6.2 Составление графика работ по проекту

Расчет прибыли и затрат на внедрение программного продукта осуществляется в случае коммерческой разработки. Поскольку данный проект

| | |
|----------------|----------------|
| Инд. № прораб. | Подпись и дата |
| Взам. инд. № | Инд. № докл. |
| Подпись и дата | |

является свободным программным обеспечением, вместо расчета прибыли необходимо составить график работ.

График работ составляется в виде диаграммы Ганта. В столбце «Наименование работы» представлены основные этапы реализации проекта, далее в столбцах следует порядок выполнения работ по дням с учетом выходных. График работ представлен на рисунке 6.2.

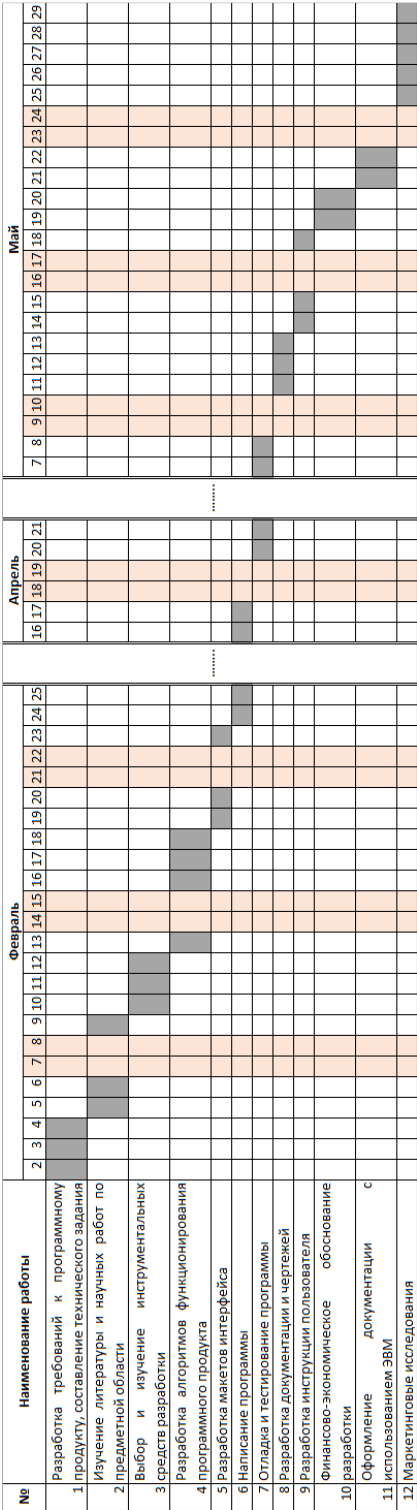


Рисунок 6.2 – График работ по проекту

Выводы

Произведен расчет затрат на разработку проекта, которые составили 138471,07 р. Срок реализации проекта по подсчетам составил 4 месяца (по 8 часов в день с учетом выходных). Поскольку разрабатываемая программа не преследует цель получения прибыли и является абсолютно бесплатной для использования в разделе не был приведен расчет цены продукта и прибыли от реализации. Расчет затрат на создание программного обеспечения можно выразить в так называемых «человеко-часах».

| | | | | | | | | | | |
|-----------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инф. № прорабл. | Подпись и дата. | Взам. инв. № | Инф. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 85 |

7 Безопасность жизнедеятельности

В данном разделе рассматриваются вопросы, связанные с безопасностью жизнедеятельности при работе с программным продуктом, а именно травмирующие, вредные и пожароопасные факторы на рабочем месте пользователя программного продукта, а также средства защиты от этих факторов. В конце раздела рассматриваются схемы системы вентиляции и общего освещения рабочего помещения на примере учебной аудитории ВятГУ.

7.1 Травмирующие, вредные и пожароопасные факторы на рабочем месте пользователя программного продукта

В результате анализа учебной аудитории 1-239 ВятГУ были выявлено несколько опасных (таблица 7.1), вредных (таблица 7.2) и пожароопасных факторов (таблица 7.3).

Таблица 7.1 – Опасные факторы при работе в аудитории 1-239 ВятГУ

| Название фактора по ГОСТ 12.0.003-74, характеристики | Единица измерения | Фактическое значение | ПДУ, ПДД, ПДК |
|--|-------------------|----------------------|---------------|
| Повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека | В | 220 | 36 |
| Острые кромки, заусенцы и шероховатость на поверхностях заготовок, инструментов и оборудования | — | — | — |

Таблица 7.2 – Вредные факторы при работе в аудитории 1-239 ВятГУ

| Название фактора по ГОСТ 12.0.003-74, характеристики | Единица измерения | Фактическое значение | ПДУ, ПДД, ПДК |
|--|-------------------|----------------------|---------------|
| Умственное перенапряжение | — | — | — |
| Перенапряжение анализаторов | — | — | — |
| — Нагрузка на органы зрения (длительность сосредоточенного наблюдения) | — | — | — |
| — Размер объекта различения | мм | 15 | 0,5 |
| Эмоциональные перегрузки | — | — | — |
| Монотонность труда | — | — | — |

| | |
|---------------|----------------|
| Инв. № дубл. | Подпись и дата |
| Взам. инв. № | |
| Инв. № прошл. | |

Продолжение таблицы 7.2

| Название фактора по ГОСТ 12.0.003-74, характеристики | | Единица измерения | Фактическое значение | ПДУ, ПДД, ПДК |
|--|--|-------------------|----------------------|---------------|
| Статическая нагрузка | | — | — | — |
| Фиксированная рабочая поза | | — | — | — |
| Вредные факторы воздушной среды | | | | |
| Повышенная или пониженная температура воздуха рабочей зоны | | — | — | — |
| — | в холодный период года для категорий работ I | °С | — | +17...+23 |
| — | в теплый период года для категорий работ I | °С | — | +18...+27 |
| Вредные факторы световой среды | | | | |
| Недостаточная освещенность рабочей зоны (общее освещение) | | лк | — | 300-500 |
| Повышенная яркость света | | кд/м ² | — | — |
| Прямая и отраженная блескость | | кд/м ² | — | — |
| Электрические поля и ЭМП | | | | |
| Напряженность электрического поля | | В/м | — | 25 |
| Плотность магнитного потока | | нТл | — | 250 |
| Напряженность электростатического поля | | кВ/м | — | 15 |

Для пользователей ЭВМ характерно:

- рабочая поза (нахождение в сидячем положении до 80% рабочего времени);
- наиболее актуальны вредные факторы, связанные с ЭМП, а также вредные факторы воздушной и световой среды.

Таблица 7.3 – Пожароопасные факторы при работе в аудитории 1-239 ВятГУ

| Горючее вещество | Тс.вос., °С | Нт., МДж/кг |
|------------------|-------------|-------------|
| Древесина | 400 | 13,8 |

Проанализировав все вредные, опасные и пожароопасные факторы и учитывая текущий технологический уровень оборудования, можно сделать вывод о том, что класс условий труда инженера-программиста является допустимым.

| | | | | |
|---------------|----------------|--------------|--------------|----------------|
| Инф. № прокл. | Подпись и дата | Взам. инб. № | Инб. № дубл. | Подпись и дата |
| | | | | |

| | | | | |
|------|------|----------|---------|------|
| Изм. | Лист | № докум. | Подпись | Дата |
| | | | | |

ТПЖА.230101.016 ПЗ

7.2 Средства защиты от травмирующих и вредных факторов

Для защиты от опасных, вредных и пожароопасных факторов, представленных в пункте 7.1, рекомендуется использовать следующие технические средства, представленные в таблице 7.4.

Таблица 7.4 – Технические средства защиты от опасных, вредных и пожароопасных факторов

| Название фактора | Технические средства защиты | Организационные средства защиты |
|--|--|--|
| Травмоопасные факторы | | |
| Повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека | Защитное зануление | Контроль изоляции; техника безопасности |
| Острые кромки, заусенцы и шероховатость на поверхностях заготовок, инструментов и оборудования | — | Техника безопасности |
| Вредные факторы | | |
| Перенапряжение анализаторов | Световые проемы; осветительные приборы | |
| Статическая нагрузка | Оборудование рабочего места | Соблюдение режима регламентированных перерывов |
| Повышенная или пониженная температура воздуха рабочей зоны | Вентиляции и очистки, кондиционирования воздуха; отопления | — |
| Недостаточная освещенность рабочей зоны (общее освещение) | Система освещения | — |
| Повышенная яркость света | Регулируемые устройства типа: жалюзи, занавеси | — |
| Прямая и отраженная блескость | Подбор материалов рабочих поверхностей | — |

| | |
|----------------|----------------|
| Инв. № дубл. | Подпись и дата |
| Взам. инв. № | |
| Подпись и дата | |
| Инв. № прошл. | |

| | | | | | | |
|------|------|----------|---------|------|--------------------|------|
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | Лист |
| | | | | | | 88 |

Продолжение таблицы 7.4

| Название фактора | Технические средства защиты | Организационные средства защиты |
|---|---|--|
| Повышенный уровень электромагнитных излучений | Подбор оборудования | Соблюдение режима регламентированных перерывов |
| Пожароопасные факторы | | |
| Пожароопасные факторы | Автоматический контроль и сигнализация; применение негорючих материалов | — |

Средства индивидуальной защиты при работе за компьютером не требуются.

7.3 Схема системы вентиляции помещения учебной аудитории

Стандарты безопасности труда предусматривают определенные нормы по обеспечению параметров воздушной среды в помещении. Для этого могут устанавливаться как системы естественной вентиляции, так и механические системы (в некоторых случаях смешанные). Вентиляция помещения также позволяет охладить воздух при избытке тепла в небольшом помещении, которым является учебная аудитория с персональными компьютерами. Свежий и прохладный воздух снижает вероятность проявления некоторых вредных психофизических факторов.

Учебная аудитория 1-239 является дисплейным классом с нормальной средой, размеры помещения – 7,5 х 6 х 3,5 м. Аудитория рассчитана на 15 человек.

Для небольшого помещения без загрязнения воздуха вредными веществами достаточно установки естественной вентиляции.

Расчет вентиляции принято начинать с определения количества подаваемого в час воздуха W_o (м³/ч). Поскольку в учебной аудитории нет вредных выделений, то воздухообмен рассчитывается по формуле

$$W_o = n \cdot V_o, \quad (7.1)$$

где n – число людей;

V_o – санитарная норма на одного человека (для объема воздуха на одного человека менее 20 м³ по принятым нормам СНиП 31-06-2009 равна 30 м³/ч).

| | | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|------|------|----------|---------|------|--------------------|----|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | 89 |

Таким образом по формуле (7.1),

$$W_o = 15 \cdot 30 = 450 \text{ м}^3/\text{ч} \quad (7.2)$$

Суммарную площадь вытяжных каналов определяют из выражения

$$S = \frac{W_o}{3600} \cdot V, \quad (7.3)$$

где W_o – необходимый воздухообмен, $\text{м}^3/\text{ч}$;

V – скорость воздушного потока в канале, $\text{м}/\text{с}$.

Скорость воздушного потока определяется как

$$V = 1,42 \cdot \mu \cdot \left(\frac{P}{\gamma_B}\right)^{-\frac{1}{2}}, \quad (7.4)$$

где μ – коэффициент, учитывающий сопротивление воздуха в канале ($\mu = 0,5$);

P – разность давлений в точке забора воздуха внутри и вне помещения, которая рассчитывается как

$$P \cong 9,8 \cdot h \cdot (\gamma_H - \gamma_B), \quad (7.5)$$

где h – разность высот между точкой приема воздуха и точкой выброса, м ;

γ_H, γ_B – плотности наружного и внутреннего воздуха

$$\gamma_x = \frac{353}{(273 + t_x)}, \quad (7.6)$$

где t_x – температура воздуха (внутреннего или наружного), $^\circ\text{C}$.

$$\gamma_H = \frac{353}{(273 + 3)} = 1,28, \quad (7.7)$$

$$\gamma_B = \frac{353}{(273 + 20)} = 1,20. \quad (7.8)$$

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 90 |

Приняв $h = 11$ м (примерная высота от уровня 2 этажа до крыши здания ВятГУ), $t_n = 3$ °С (как средняя температура по данным метеослужб в г. Кирове за учебный период с сентября по июнь), $t_e = 20$ °С (по нормам СанПин 2.4.2.1178.02), можно рассчитать разность давлений по формуле (7.5)

$$P \cong 9,8 \cdot 11 \cdot (1,28 - 1,20) = 8,624 \text{ Па.} \quad (7.9)$$

Отсюда, скорость воздушного потока по формуле (7.4)

$$V = 1,42 \cdot 0,5 \cdot \left(\frac{8,624}{1,20} \right)^{-\frac{1}{2}} = 0,265 \text{ м/с.} \quad (7.10)$$

Суммарная площадь вытяжных каналов по формуле (7.3)

$$S = \frac{450}{3600} \cdot 0,265 = 0,033 \text{ м}^2. \quad (7.11)$$

Приняв размеры вытяжного канала 150x150мм можно рассчитать число вытяжных каналов

$$n = \frac{S_{\text{общ}}}{S_{\text{кан}}}, \quad (7.12)$$

где $S_{\text{общ}}$ – суммарная площадь вытяжных каналов;

$S_{\text{кан}}$ – площадь сечения вытяжного канала.

Отсюда по формуле (7.12)

$$n = \frac{0,033}{0,0225} = 1,46 \approx 2. \quad (7.13)$$

Следовательно, двух вытяжных каналов размером 150x150мм достаточно для обеспечения естественной вентиляции помещения аудитории. Вытяжные каналы устанавливаются как правило на расстоянии 0,5м и более от плоскости потолка. Эскиз плоскости стены, в которой устанавливаются вытяжной канал представлен на рисунке 7.1. Данная стена расположена напротив окон помещения.

| | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | |
| | | | | | 91 | | | | |

Количество рядов светильников в помещении

$$n = \frac{B}{L_B + l_{\text{свет}}}, \quad (7.14)$$

где В – ширина помещения;

L_B – расстояние между рядами светильников (1,2м – рекомендуемое значение для (КСС)-Д);

$l_{\text{свет}}$ – длина светильника (0,595м).

По формуле (7.14)

$$n = \frac{6}{1,2 + 0,595} = 3,34 \approx 3 \text{ шт.} \quad (7.15)$$

Расстояние по ширине от крайнего светильника до стены

$$l_b = \frac{B - (n - 1) \cdot L_B - n \cdot l_{\text{свет}}}{2}, \quad (7.16)$$

где n – количество рядов светильников.

По формуле (7.16)

$$l_b = \frac{6 - (3 - 1) \cdot 1,2 - 3 \cdot 0,595}{2} = 0,908 \text{ м.} \quad (7.17)$$

Расстояние от светильника до рабочей поверхности

$$h = H - h_{\text{раб}} - h_c, \quad (7.18)$$

где Н – высота помещения, м;

$h_{\text{раб}}$ – высота рабочей поверхности принимается по таблице К.1 СНиП 23-05-95 (0,8м);

h_c – высота света светильника.

По формуле (7.18)

| | | | | | | | | | | |
|----------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прайдл. | Подпись и дата. | Взам. инв. № | Инв. № дудл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 93 |
| Изм. | Лист | № докум. | Подпись | Дата | | | | | | |

$$h = 3,5 - 0,8 - 0 = 2,7 \text{ м.} \quad (7.19)$$

Индекс помещения

$$i = \frac{A \cdot B}{h \cdot (A + B)}, \quad (7.20)$$

где А, В и h – длина и ширина и расстояние от светильника до рабочей поверхности

По формуле (7.20)

$$i = \frac{7,5 \cdot 6}{2,7 \cdot (7,5 + 6)} = 1,23 \quad (7.21)$$

Расчетное количество ламп

$$N = \frac{E_H \cdot k \cdot S \cdot Z}{\Phi_{\Pi} \cdot n \cdot \eta}, \quad (7.22)$$

где E_H – нормативная минимальная освещенность, лк;

k – коэффициент запаса (для кабинета информатики $K_3 = 1,2$);

Z – коэффициент минимальной освещенности (для люминесцентных ламп при расположении светильников в ряд рекомендуется принимать $Z = 1,1$);

S – площадь помещения, м^2 ;

n – число рядов светильников;

Φ_{Π} – световой поток одного светильника (1080 лм);

η – коэффициент использования светового потока (для заданных коэффициентов отражения и расчетного индекса помещения принимается равным 0,51).

По формуле (7.22)

$$N = \frac{400 \cdot 1,2 \cdot (7,5 \cdot 6) \cdot 1,1}{1080 \cdot 3 \cdot 0,51} = 14,3 \text{ шт.} \quad (7.23)$$

Для 3 рядов с равным количеством ламп принимается общее количество ламп 12 шт (по 4 лампы на ряд). Расстояние по длине от крайнего светильника до стены

| | | | | | | | | | |
|----------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|
| Инв. № прайдл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | |
| | | | | | 94 | | | | |

$$l_a = \frac{A - (n_p - 1) \cdot L_A - n_p \cdot l_{\text{свет}}}{2}, \quad (7.24)$$

где A – длина помещения;

n_p – количество светильников в одном ряду;

L_A – расстояние между светильниками (1,2м – рекомендуемое значение для (КСС)-Д)

По формуле (7.24)

$$l_a = \frac{7,5 - (4 - 1) \cdot 1,2 - 4 \cdot 0,595}{2} = 0,76 \text{ м.} \quad (7.25)$$

Эскиз схемы освещения аудитории представлена на рисунке 7.2.

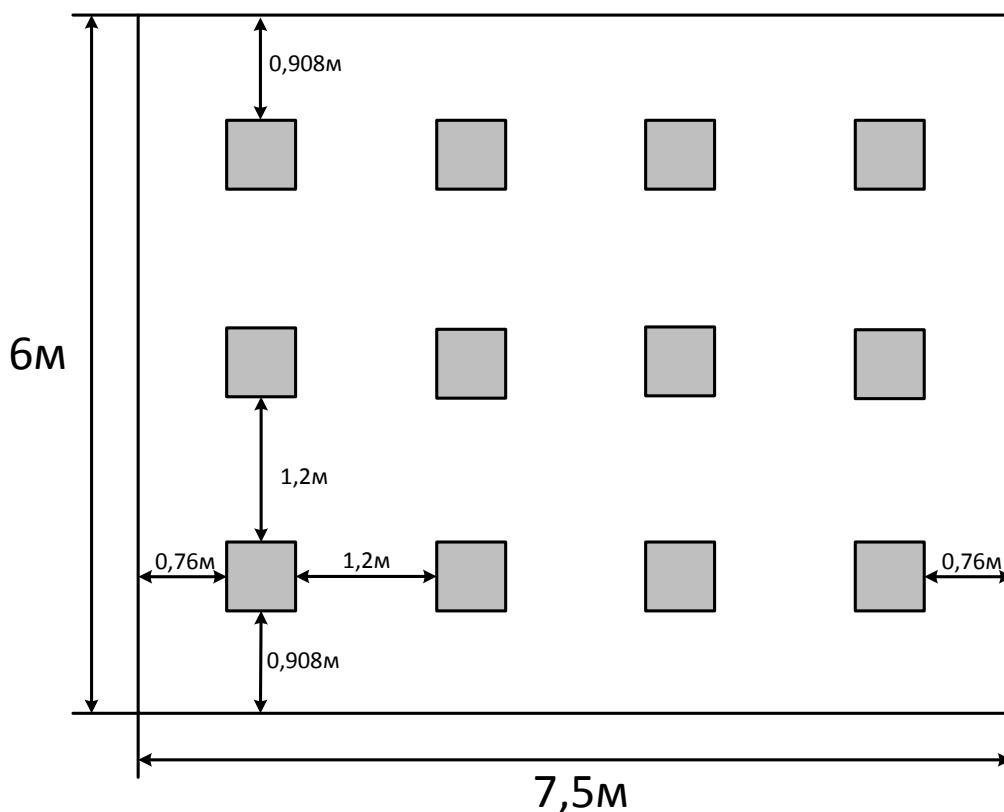


Рисунок 7.2 – Эскиз схемы освещения учебной аудитории

Выводы

Был проведен анализ опасных, вредных и пожароопасных факторов при работе в аудитории ВятГУ 1-239. Данная аудитория является дисплейным классом с несколькими персональными компьютерами. Выявлено, что наиболее вредными

| | |
|----------------|--|
| Подпись и дата | |
| Инв. № дубл. | |
| Взам. инв. № | |
| Подпись и дата | |
| Инв. № прошл. | |

| | | | | |
|------|------|----------|---------|------|
| | | | | |
| Изм. | Лист | № докум. | Подпись | Дата |

ТПЖА.230101.016 ПЗ

факторами при работе являются факторы воздушной и световой среды, а также связанные с ЭМП. Однако, учитывая текущий технологический уровень используемого оборудования, работу за компьютером можно отнести к классу допустимых.

Было выявлено, что для помещения без загрязнений воздуха вредными веществами достаточно установки системы естественной вентиляции. В связи с этим произведен расчет необходимого воздухообмена для аудитории, на основе которого были рассчитаны размеры вентиляционных отверстий. Для данного помещения достаточно двух вентиляционных отверстий размером 150x150 мм.

На основании того, что для помещения, в котором нет крупных затеняющих объектов, необходимо равномерное горизонтальное освещение, расчет схемы освещения производился методом коэффициента использования светового потока. Данный метод позволил рассчитать необходимую мощность ламп и их количество. Исходя из требований СНиП 23-05-95 о нормативной минимальной освещенности аудиторий высших учебных заведений, выявлено, что для данной аудитории необходимо 12 ламп (3 ряда по 4 лампы) с типом светильника ЛВО-13 и со световым потоком 1080 лм.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 96 |

Заключение

В ходе дипломного проектирования была разработана программа дедуктивного и абдуктивного логического вывода с построением схемы. Основной целью разработки программы являлась поддержка исследований в области логического вывода делением дизъюнктов путем автоматизации процесса решения логических задач с помощью дедуктивного и абдуктивного вывода с построением схемы. Программа предназначена для экспериментальных исследований методов логического вывода делением дизъюнктов, выявления возможных исключительных ситуаций при решениях тех или иных логических задач, а также поиска новых классов задач, в которых может применяться логический вывод.

Получены следующие результаты дипломного проектирования:

- разработан метод абдуктивного логического вывода заключения с построением схемы, отличающийся от известных методов применением процедуры обобщенного деления дизъюнктов, а также тем, что с помощью специальной процедуры полученные в результате абдуктивного вывода дополнительные посылки преобразуются в дополнительные элементы схемы логического вывода;

- разработаны программные алгоритмы, послужившие основой для проектирования структуры библиотеки логического вывода и сопутствующих программных модулей, а также упрощающие задачу написания программного кода;

- разработана программная библиотека логического вывода, содержащая программы реализации дедуктивного и абдуктивного логического вывода с построением схемы. Данная библиотека не имеет зависимостей от текущего проекта и может быть повторно использована для других проектов по данной тематике. В библиотеке представлен механизм многопоточного выполнения процедур логического вывода. При этом не рассматривается эффективность многопоточной реализации. Экспериментально было установлено, что при решении небольших задач логического вывода в логике высказываний многопоточное выполнение процедур не дает ощутимого прироста производительности. Это связано в первую очередь с тем, что процедуры логического вывода в логике высказываний выполняются достаточно быстро и в основном машинное время затрачивается на инициализацию потоков и переключение контекста в многопоточной среде. Однако, стоит отметить, что процедуры логического вывода в логике предикатов выполняются существенно дольше и здесь ключевую роль в вопросе производительности может сыграть их параллельное выполнение.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | Лист |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | 97 |

– разработан декларативный язык логического программирования, позволяющий описать исходную базу знаний в виде правил логики высказываний. Язык имеет достаточно простой синтаксис, который будет привычен пользователям, знакомым с программированием. Правила можно ввести, используя в основном только буквы латинского алфавита, без необходимости частого ввода специальных символов.

– разработана библиотека построения схемы логического вывода, основанная на компонентах технологии JavaFX. При этом, стоит отметить, что библиотека не имеет зависимостей от каких-либо сущностей или методов других модулей разрабатываемой программы. Таким образом она может быть использована в других проектах для построения интерактивных графов, используя при этом в качестве узлов любой графический компонент платформы JavaFX;

– разработан интерфейс пользователя для программы, позволяющий создать проект базы знаний или открыть существующий, выполнить ввод исходных правил с помощью разработанного декларативного языка, выбрать метод логического вывода для решения определенной задачи, сохранить созданный проект в файл, при этом выполнив шифрование внутреннего представления файла.

Дальнейшее развитие разработанной программы может проводиться по нескольким направлениям:

– реализация работы программной библиотеки логического вывода с правилами, описанными в логике предикатов;

– расширение возможностей интерпретатора разработанного декларативного языка: упрощение сложных правил с применением законов булевой алгебры, а также работа с правилами, описанными в логике предикатов;

– разработка модуля логического прогнозирования, позволяющего на основе результатов логического вывода выполнять анализ состояний какого-либо исследуемого объекта, осуществлять прогноз будущих состояний объекта, определять наличие множества вариантов путей перехода в определенное состояние.

– реализация многопоточного механизма работы интерфейса программы, при котором сложные задачи, требующие большое количество времени на выполнение, запускались бы как фоновые процессы, не блокирующие интерфейс пользователя.

| | | | | | | | | | | |
|---------------|-----------------|--------------|--------------|----------------|--------------------|--|--|--|--|------|
| Инв. № прокл. | Подпись и дата. | Взам. инв. № | Инв. № дубл. | Подпись и дата | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Изм. | Лист | № докум. | Подпись | Дата | ТПЖА.230101.016 ПЗ | | | | | Лист |
| | | | | | | | | | | 98 |