Санкт-Петербургский государственный политехнический университет

Институт информационных технологий и управления

Кафедра компьютерных систем и программных технологий

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: Разработка синтаксического анализатора для библиотеки логического вывода на базе дизъюнктов Хорна

Направление: 230100 – Информатика и вычислительная техника

|  |  |
| --- | --- |
| Выполнила студентка | Галустова П.А. |
| Руководитель к.т.н. доцент | Глухих М.И. |

Санкт-Петербург

2015

РЕФЕРАТ

N страниц, n рисунков, n приложений

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР, ПАРСЕР, ЛОГИЧЕСКИЙ ВЫВОД, ПРОЛОГ

В работе выполнена разработка синтаксического анализатора для существующей библиотеки логического вывода. Также разработан пользовательский интерфейс, позволяющий использовать эту библиотеку для осуществления логического вывода на практике. Проведено тестирование разработанного синтаксического анализатора на различных примерах, иллюстрирующих как правильные, так и ошибочные запросы.

Оглавление

[**Введение** 4](#_Toc422091554)

[**Глава 1.** **!** 5](#_Toc422091555)

[**1.1.** **Язык PROLOG** 5](#_Toc422091556)

[**1.2.** **Методы доказательства логических утверждений** 6](#_Toc422091557)

[**1.3.** **Описание библиотеки логического вывода** 7](#_Toc422091558)

[**Глава 2.** **РАЗРАБОТКА ПРОГРАММНОЙ РЕАЛИЗАЦИИ** 9](#_Toc422091559)

[**2.1** **Разработка синтаксиса входных данных** 9](#_Toc422091560)

[**2.2** **Описание разработанного модуля синтаксического анализа** 11](#_Toc422091561)

[**2.3** **Описание пользовательского интерфейса** 14](#_Toc422091562)

[**Глава 3.** **ПРАКТИЧЕСКИЕ РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ** 16](#_Toc422091563)

[**3.1.** **Простые примеры** 16](#_Toc422091564)

[**3.2.** **Пример с последовательным сравнением** 16](#_Toc422091565)

[**3.3.** **Пример с математическими операциями и сравнением** 17](#_Toc422091566)

# Введение

**Актуальность работы.** Актуальность данной работы состоит в том, что для исходной библиотеки логического вывода не существует возможности ввода пользовательских данных

**Цель работы.** Разработка синтаксического анализатора для библиотеки логического вывода на базе дизъюнктов Хорна.

**Решаемые задачи.**

1. Разработка формата входных данных
2. Разработка программной реализации синтаксического анализатора
3. Разработка механизма подключения правил
4. Разработка интерфейса взаимодействия с пользователем

**Практические результаты.** В ходе данной работы был создан синтаксический анализатор, позволяющий использовать библиотеку логического вывода на практике.

**Структура работы.** Пояснительная записка кбакалаврской работе состоит изтрех глав.

**Первая глава** бакалаврской работы посвящена рассмотрению основных понятий и теоретических данных, лежащих в основе исходной программы. В частности, рассматривается понятие дизъюнктов Хорна, устройство языка PROLOG, понятие логического вывода и возможные методы его осуществления.

Во **второй главе** представлена программная реализация синтаксического анализатора и пользовательского интерфейса. Описывается схема работы программы.

В **третьей главе** проводится тестирование разработанной программы на различных примерах, иллюстрирующих как правильные, так и ошибочные запросы. На основании тестов делается вывод о выполнении технического задания.

В **заключении** формулируются основные результаты работы, направления дальнейшего развития.

1. **!**
   1. **Язык PROLOG**

Язык программирования Пролог (PROgramming LOGic) предполагает получение решения задачи при помощи логического вывода из ранее известных фактов. Программа на языке Пролог не является последовательностью действий – она представляет собой набор фактов и правил, обеспечивающих получение логических заключений из данных фактов. Поэтому Пролог считается декларативным языком программирования. Пролог базируется на фразах (предложениях) Хорна, являющихся подмножеством формальной системы, называемой логикой предикатов. В математической логике, фраза или дизъюнкт Хорна — дизъюнкция литералов с не более чем одним положительным литералом. Литерал — это атомарная формула или её логическое отрицание. Дизъюнкт Хорна без положительных литералов иногда называется целью или запросом, в частности в логическом программировании.

Одной из важнейших особенностей Пролога является то, что он ищет не только ответ на поставленный вопрос, но и все возможные альтернативные решения. Вместо обычной работы программы на процедурном языке от начала и до конца, Пролог может возвращаться назад и просматривать все остальные пути при решении всех частей задачи. Общий принцип выполнения программ на Прологе прост: производится поиск ответа на вопросы, задаваемые БД, состоящей из фактов и правил, то есть проверяется соответствие предикатов вопроса предложениям из БД. Это частный случай метода резолюций.

Основными понятиями в языке Пролог являются факты, правила логического вывода и запросы, позволяющие описывать базы знаний, процедуры логического вывода и принятия решений. Факт является простым предикатом, который записывается в виде функционального терма, состоящего из имени отношения и объектов, заключенных в круглые скобки. Вторым типом предложений Пролога является вопрос или цель. Цель – это средство формулировки задачи, которую должна решать программа. Третьим типом предложения является правило. Правило позволяет вывести один факт из других фактов. Иными словами, правило – это заключение, для которого известно, что оно истинно, если одно или несколько других найденных заключений или фактов являются истинными.

* 1. **Методы доказательства логических утверждений**

Вывод в логической системе является процедурой, которая из заданной группы выражений выводит отличное от заданных семантически правильное выражение. Эта процедура, представленная в определенной форме, и является правилом вывода. Если группа выражений, образующая посылку, является истинной, то должно гарантироваться, что применение правила вывода обеспечит получение истинного выражения в качестве заключения.

Наиболее часто используются два метода:

* **Прямой вывод.** В прямом выводе используется знание семантики тех операторов, через которые строятся аксиомы. Так, если аксиома утверждает, что A&B, то из смысла этого утверждения следует, что истинными будут высказывния A и B, которые войдут в цепочку вывода. Если известно, что истинным являются высказывания {AvB, ‾A}, то истинным будет высказывание B именно исходя из смысла этих высказываний. В прямом выводе строится цепочка высказываний, которая и является выводом.
* **Метод резолюций.** Метод резолюций опирается на исчисление резольвент. Существует теорема, утверждающая, что вопрос о доказуемости произвольной формулы в исчислении предикатов сводится к вопросу о доказуемости пустого списка в исчислении резольвент. Поэтому доказательство того, что список формул в исчислении резольвент пуст, эквивалентно доказательству ложности формулы в исчислении предикатов.

Для порождения логических следствий используется очень простая схема рассуждений. Пусть А, В, X – формулы. Предположим, что две формулы (A∨X) и (B∨¬X) –истинны. Если X –истинна, то следовательно В истинна. Наоборот, если X ложна, то можно заключить, что А- истинна. В обоих случаях (A∨ В) истинна. Получается правило

{*A*∨*X*, *B*∨¬*X*}⎥= *A* ∨ В

Это правило называется правилом резолюций.

Алгоритмы доказательства выводимости A \vdash B, построенные на основе этого метода, применяются во многих системах искусственного интеллекта, а также являются фундаментом, на котором построен язык логического программирования «[Пролог](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BB%D0%BE%D0%B3_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))».

* 1. **Описание библиотеки логического вывода**

Данная библиотека предназначена для доказательства или опровержения заданного условия на основе правил вывода, а также фактов, задаваемых пользователем.

Основными элементами синтаксиса являются предикаты, которые по своему внешнему виду похожи на факты в Прологе, они также участвуют в образовании правил. Для хранения предикатов-фактов предназначен класс *Theory*, в него, при помощи функции *addPredicate,* добавляются элементы типа *Predicate*. Для хранения и дальнейшего использования правил предназначен класс *Logician*, правила описаны в классе InferenceRule и добавляются при помощи функции addRule.

Для осуществления логического вывода необходимо выполнить следующие этапы:

1. Создание экземпляра класса Logician и добавление к нему исходных фактов
2. Создание экземпляра класса Theory и подключение к нему правил логического вывода
3. Осуществление логического вывода при помощи функций экземпляра класса Logician – proveTrue / proveFalse.
4. **РАЗРАБОТКА ПРОГРАММНОЙ РЕАЛИЗАЦИИ**

В рамках данной главы решается задача программной реализации синтаксического анализа и дальнейшего использования входных данных.

В качестве языка программирования используется язык Java, среда разработки NetBeans IDE 8.0.2.

* 1. **Разработка синтаксиса входных данных**

Одной из основных задач при разработке синтаксического анализатора стоял выбор формата ввода входных данных. При выборе формата необходимо было сохранять равновесие между удобством и понятностью текста для пользователя и возможностью распознавания во входных данных рабочих структур программой.

Выбранный формат данных аналогичен синтаксису языка ПРОЛОГ. Его преимущество с программной стороны состоит в том, что он достаточно прост для реализации, так как используемые конструкции имеют похожий вид и могут быть описаны, например, с помощью регулярных выражений. Также все используемые имена могут быть однозначно трактованы, как английские слова, обозначающие ту или иную операцию над переменными. В связи с тем, что в библиотеке логического вывода тип PredicateType задан, как enum, также удобно использовать в качестве имен предикатов имена, аналогичные элементам этого класса.

Ниже приведено более подробное описание выбранного формата входных данных.

Входные данные подаются в программу при помощи .txt файлов или текста из редактора, содержащих предикаты-факты, и предикат-цель.

* Предикат-факт имеет вид:

NAME(аrg1, arg2,…)

, где  *NAME* – это имя предиката, а *аrg1, arg2*– аргументы. Аргументы в предикате могут быть либо переменными (записываются строчными буквами), либо целочисленными константами.

* Правило имеет вид:

TARG(arg1, arg2,…) :- NAME(аrg1, arg2,…), NAME(аrg1, arg2,…), …

, где слева от разделителя «:-» стоит предикат-вывод, а справа – предикаты-условия. В качестве аргументов могут фигурировать: целочисленные константы, константа (прописные буквы), переменная (строчные буквы).

* Предикат-цель имеет вид:

?NAME(аrg1, arg2,…)

или

?(!)NAME(аrg1, arg2,…)

, где наличие «!» обозначает, что необходимо доказать ложность утверждения, а отсутствие – истинность.

Для определения принадлежности строки к одному из трёх типов используется функция matches() из библиотеки String, которая позволяет проверить, соответствует ли строка заданному регулярному выражению. Для определения использовались следующие регулярные выражения:

String regexArg = "((\_{0,1}[a-z]+)|(\\-{0,1}[0-9]+)|([A-Z]+))"; (1)

String regexFact = "[A-Z]+(\_[A-Z]+)\*\\(" + regexArg + "(,"+regexArg +")\*\\)"; (2)

String regexRule = regexFact + ":-" + regexFact + "(," + regexFact + ")\*"; (3)

String regexTargetTrue = "\\?" + regexFact; (4)

String regexTargetFalse = "\\?!" + regexFact; (5)

Регулярное выражение (1) описывает формат возможных аргументов, то есть аргумент может начинаться с подчеркивания и записываться строчными буквами, может быть целым числом со знаком или без или может быть записан прописными буквами.

Регулярное выражение (2) описывает формат предиката, то есть предикат начинается с имени, записанного прописными буквами, которое может содержать подчеркивания, далее в скобках через запятую идут аргументы, которых может быть один или несколько.

Регулярное выражение (3) описывает формат правила. Правило может быть условно поделено на две части, аналогично синтаксису языка ПРОЛОГ. Слева записывается предикат-вывод, справа минимум один предикат-условие.

Регулярные выражения (5) и (6) описывают формат предиката-цели. Они различаются наличием символа «!», который обозначает необходимость доказать ложность заданного утверждения.

* 1. **Описание разработанного модуля синтаксического анализа**

Для осуществления поставленных задач был разработан класс *Parser.java.*

Действия, осуществляемые данным классом можно условно разбить на три этапа:

1. Построчное чтение входных данных и определение вида каждой строки

За эти действия отвечают функции parseFile и parseText. Они имеют похожую внутреннюю структуру, которая представлена на рис. 2.2.1. При обработке каждой строки производится ее сравнение с регулярным выражением, что позволяет проверить правильность написания и установить её вид, т.е. является ли она целью или фактом.

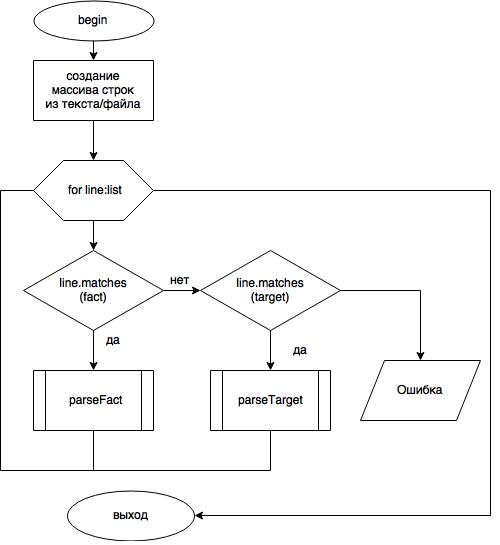


Рис. 2.2.1. Блок-схема функций parseFile и parseText

1. В соответствии с видом строки вызов функций, предназначенных для распознавания строки в объект типа Predicate. Эти действия осуществляются в функциях parseFact и parseTarget. Структура функции parseFact представлена на рис. 2.2.2.

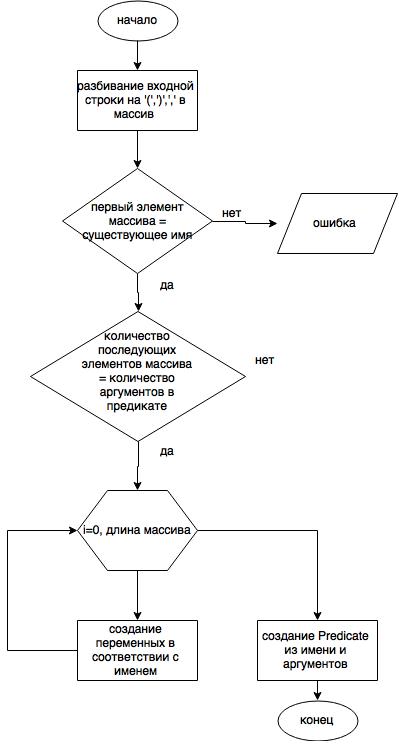


Рис. 2.2.2. Блок схема функций parseFact

Функция parseTarget осуществляет удаление из строки символов «?» и «!», которые обозначают то, что эта строка обозначает предикат-цель, после чего вызывается функция parseFact с полученным тексом в качестве аргумента.

1. Добавление правил. Это действие осуществляется в функции addRules, которая практически аналогична функции parseFile. К каждой строке входного файла применяется регулярное выражение, позволяющее установить правильность записи правила. В случае если ошибки не обнаружено, строка разбивается на факты, каждый из которых обрабатывается в функции parseItemQ, являющейся аналогом функции parseFact.
   1. **Описание пользовательского интерфейса**

Разработанный интерфейс представляет собой текстовый редактор, обладающий функциями загрузки и сохранения файлов, а также поддерживающий функцию доказательства теорем. Внешний вид приложения представлен на рис. 2.3.1.

Меню «File» содержит пункты «Save» и «Load», осуществляющие сохранение и загрузку файла соответственно.

Меню «Logic» содержит пункт «Prove», запускающий процесс обработки текста в окне редактора и последующий запуск процесса доказательства с выводом результата в диалоговом окне, а также пункт «Rules», открывающий меню выбора используемых правил. Последний пункт позволяет выбрать все стандартные правила (правила сравнения, работы с указателями, арифметики), а также выбрать собственный файл с правилами. Вид окна настроек представлен на рисунке 2.3.2.

Также, в окне доступны кнопки «Load» и «Prove», которые дублируют одноименные пункты меню и «Clear», нажатие на которую очищает поле редактора.

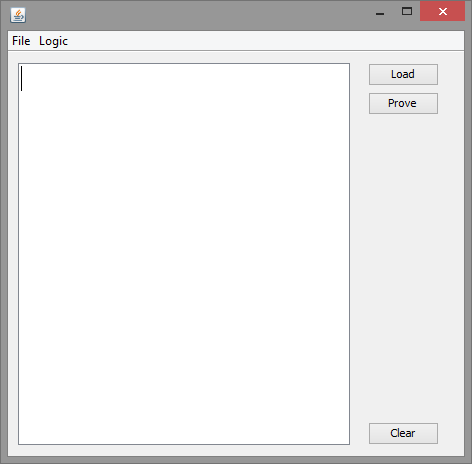


Рис. 2.3.1. Пользовательский интерфейс

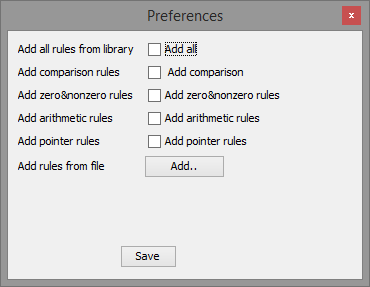


Рис. 2.3.2 Окно настроек правил

1. **ПРАКТИЧЕСКИЕ РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ**

В рамках данной главы проводится экспериментальное исследование разработанного модуля на различных тестовых данных. Также протестирован механизм подключения стандартных и пользовательских правил.

* 1. **Простые примеры**

Следующие два примера можно назвать простыми, так как для их доказательства не требуется подключение правил.

**Дано:**

**Доказать:**

**Данные вводимые в программу:**

LESS(x,y)

?LESS(x,y)

**Результат работы программы:**

The result is true

**Дано:**

**Доказать:**

– ложь

**Данные вводимые в программу:**

LESS(x,y)

?!LESS (y,x)

**Результат работы программы:**

The result is true

* 1. **Пример с последовательным сравнением**

Без подключения правил доказать истинность предиката-цели в следующем примере невозможно, поэтому необходимо подключить правила сравнения через пункт меню «Logic» -> «Rules» (см. рис 3.2.1)

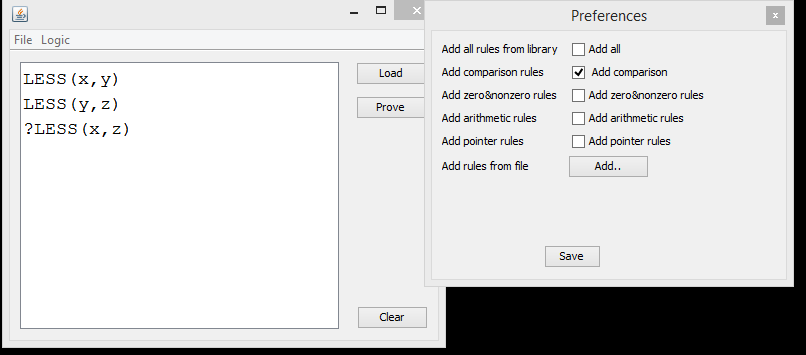


Рис. 3.2.1. Подключение правил сравнения

**Дано:**

,

**Доказать:**

**Данные вводимые в программу:**

LESS(x,y)

LESS(y,z)

?LESS(x,z)

**Результат работы программы:**

The result is true

* 1. **Пример с математическими операциями и сравнением**
  2. **Пример подключения пользовательских правил**