Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 4

по курсу «Экспертные системы»

на тему: «Метод резолюции в логике предикатов первого порядка»

Студент ИУ7-31М (Группа) _____ (Подпись, дата) Клименко А. К. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Русакова З. Н. (И. О. Фамилия)

2024 г.

Введение

Цель лабораторной работы — приобретение практических навыков реализации алгоритма обратного дедуктивного вывода в определенных выражениях.

Задачи работы:

1. Разработать алгоритм обратного дедуктивного вывода для логики предикатов первого порядка с использованием функциональных символов.
2. Разработать программу, реализующую разработанный алгоритм для произвольного множества определенных выражений.
3. Протестировать программу для различных логических задач.

1 Теоретический раздел

1.1 Логика высказываний

Повествовательное предложение, о котором можно сказать истинно оно или ложно, называется **высказываем**. Для высказываний вводятся буквы, которые называются атомами или пропозициональными переменными. Из простых высказываний с помощью логических связок строятся сложные высказывания.

Общезначимая формула — формула, истинная при всех интерпретациях. Остальные формулы **выполнимы**, если они истины в какой-либо интерпретации.

Для решения практических задач необходимо приводить все формулы в КНФ. Алгоритм приведения формулы логики высказываний к КНФ:

1. Устранить связки импликаций и эквиваленций используя соотношения:

$$A \to B = \neg A \vee B = \neg(A \& \neg B)$$
$$A \leftrightarrow B = (A \to B) \& (B \to A)$$

2. Продвинуть отрицания до атомов используя законы де Моргана:

$$\neg(A \& B) = \neg A \vee \neg B$$
$$\neg(A \vee B) = \neg A \& \neg B$$

3. Применить закон дистрибутивности:

$$A \vee (B \& C) = (A \vee B) \& (A \vee C)$$

Формула F является **логическим следствием** n формул A_i, если она истина во всех интерпретациях, в которых истины все A_i. Из этого вытекает, что импликация конъюнкций A_i к F общезначима:

$$(A_1 \& A_2 \& \ldots \& A_n) \to F$$

Общезначимость не доказывается, так как это требует рассмотрения

всех интерпретаций. Поэтому переходят к доказательству противоречивости отрицания $\neg F$. Формула представляется как конъюнкция дизъюнктов. Если хоть один дизъюнкт будет ложным — то мы доказали противоречие. В основе доказательства лежит процедура резолюции, которая вводит понятие резольвенты двух дизъюнктов. Если у нас есть два дизъюнкта, в которые входит один и тот же атом, но с разными знаками (эти атомы называются контрарной парой в дизъюнктах), то можем получить дизъюнкт, который является объединением атомов первого и второго дизъюнкта за исключением контрарной пары. Этот дизъюнкт называется резольвентой и легко показать, что он является логическим следствием двух дизъюнктов.

Алгоритм вывода по методу резолюции:

1. Принять отрицание заключения.
2. Привести все формулы посылок (аксиом) и отрицание цели к КНФ.
3. Выделить список дизъюнктов.
4. Если существует пара дизъюнктов, содержащих контрарные атомы, то эти дизъюнкты объединяются с удалением контрарной пары и получаем новый дизъюнкт — резольвенту, которая является логическим следствием и добавляется к исходному множеству дизъюнктов.

Исходный алгоритм анализа дизъюнктов — полный перебор. Если на каком-то шаге мы получим два дизъюнкта по типу $A, \neg A$, резольвента будет пустым дизъюнктом или ложью. Если не получили пустого дизъюнкта — продолжаем перебор на новом множестве.

Нулевой пример:

$$\frac{A \vee B, \quad (A \to C), \quad B \to D}{C \vee D}$$

1. $\neg(C \vee D) = (\neg C \wedge \neg D)$
2. $(A \to C) = (\neg A \vee C)$
3. $(B \to D) = (\neg B \vee D)$
4. $K = (A \vee B, \neg A \vee C, \neg B \vee D, \neg C, \neg D)$

5. $[A \vee C + \neg C] : K = (A \vee B, \neg A, \neg B \vee D, \neg D)$

6. $[A \vee B + \neg A] : K = (B, \neg B \vee D, \neg D)$

7. $[B + \neg B \vee D] : K = (D, \neg D)$

8. $K = ()$

Реализовали стратегию унитарности, когда искали дизъюнкт с наименьшим количеством атомов. В общем случае построение резольвент выполняется полным перебором для текущего множества дизъюнктов, однако существуют модификации этого алгоритма:

1. **Стратегия унитарности** — выбирать самый короткий дизъюнкт.

2. **Линейная резолюция** — задаем исходное множество дизъюнктов, получаем резольвенту для пары дизъюнктов из него. Один из дизъюнктов целесообразно взять из отрицания цели. На следующем шаге искать резольвенту, где один дизъюнкт — последняя резольвента, второй дизъюнкт из исходного множества. Либо один дизъюнкт должен быть получен как резольвента.

3. **Идея опорного множетсва**. Исходное множество дизъюнктов разделяют на множество (исходное), которое относится к аксиомам, и множество (опорное), полученное из отрицания цели. Резольвенты формируются один дизъюнкт из опорного множетсва, другой из исходного и полученных резольвент.

1.2 Логика предикатов первого порядка

Логика предикатов поволяет описывать как свойства, так и отношения между объектами. Под **предикатом** будем принимать повествовательное предложение, которое содержит переменные и при означивании переменных константами становится высказыванием, о котором можно сказать истинно оно или ложно.

Предикат на некотором множетсве W есть логическая функция, которая при означивании аргументов превращается в высказывание со значениями

истина или ложь. Эта логическая функция может зависеть от одной переменной, от двух или от n переменных. В случае одной переменной — это отражение свойства объекта. В случае n переменных — это отражение связи между n объектами. Поэтому говоря о предикатах, необходимо задать интерпретацию — множество объектов, на которых предикат определен. В случае предиката от одной переменной, переменная принимает значения объекта из фиксированной области.

Также вводятся функци от n объектов, которые отображают n объектов в 1 объект из наших областей. Константа и переменная называются термом. Функция от термов — тоже терм. $f(t_1, t_2, t_3)$ — элементарная формула.

Следующий тип формул — использование кванторов всеобщности и существования. Роли переменных связывают их с предметной областью. Формулы логики предикатов: Квантор общности — для предиката с одной переменной означает, что он истиннен для всей области определения. Множество истинности предиката — те значения переменных, на которых предикат становится истинным. Квантор существования означает, что есть хотя бы один объект, для которого это свойство истинно. Если мы имеем предикат от 1 переменной, то накладывая квантор на него мы получаем высказывание.

Рассмотрим n-местные предикаты. Часть переменных может находиться под кванторами. Эти переменные называются связанными. Остальные — свободными. А арность нового предиката становится уже меньше.

Правила проноса отрицания для кванторов:

$$\neg\forall x P(x) = \exists x \neg P(x)$$
$$\neg\exists x P(x) = \forall x \neg P(x)$$

2 Практический раздел

2.1 EBNF грамматика выражений логики предикатов

Ниже приведена грамматика выражений логики предикатов, реализованная в данной лабораторной работе.

```
<expr>       ::= <equ>
<equ>        ::= <disj> <eq-sign> <equ>
<disj>       ::= <impl> '+' <disj>
<impl>       ::= <conj> '->' <impl>
<conj>       ::= <quant> '&' <conj>
<quant>      ::= [ <quant-list> ] <term>
<term>       ::= '(' <expr> ')' | <neg> | <atom>
<neg>        ::= '~' <quant>
<equ-sign>   ::= '==' | '=' | '<=>' | '<->'
<atom>       ::= IDENT [ '(' <arg-list> ')' ]
<arg-list>   ::= <arg> [ ',' <arg-list> ]
<arg>        ::= IDENT [ '(' <arg-list> ')' ]
<quant-list> ::= <quantifier> [ <quant-list> ]
<quantifier> ::= '\forall' '(' <var-decl> ')'
                | '\exists' '(' <var-decl> ')'
<var-decl>   ::= IDENT [ ',' <var-decl> ]
```

В качестве примера, формула

$$\forall x \forall y(\neg P(x, f(C)) \rightarrow \exists z(Q(x, z) \& P(g(x), y)))$$

по правилам представленной грамматики записывается в следующем виде

```
\forall(x, y) (~P(x, f(C)) -> \exists(z) (Q(x, z) & P(g(x), y)))
```

2.2 Алгоритм унификации двух атомов

Схема алгоритма унификации двух атомов представлена на рисунке 2.1.

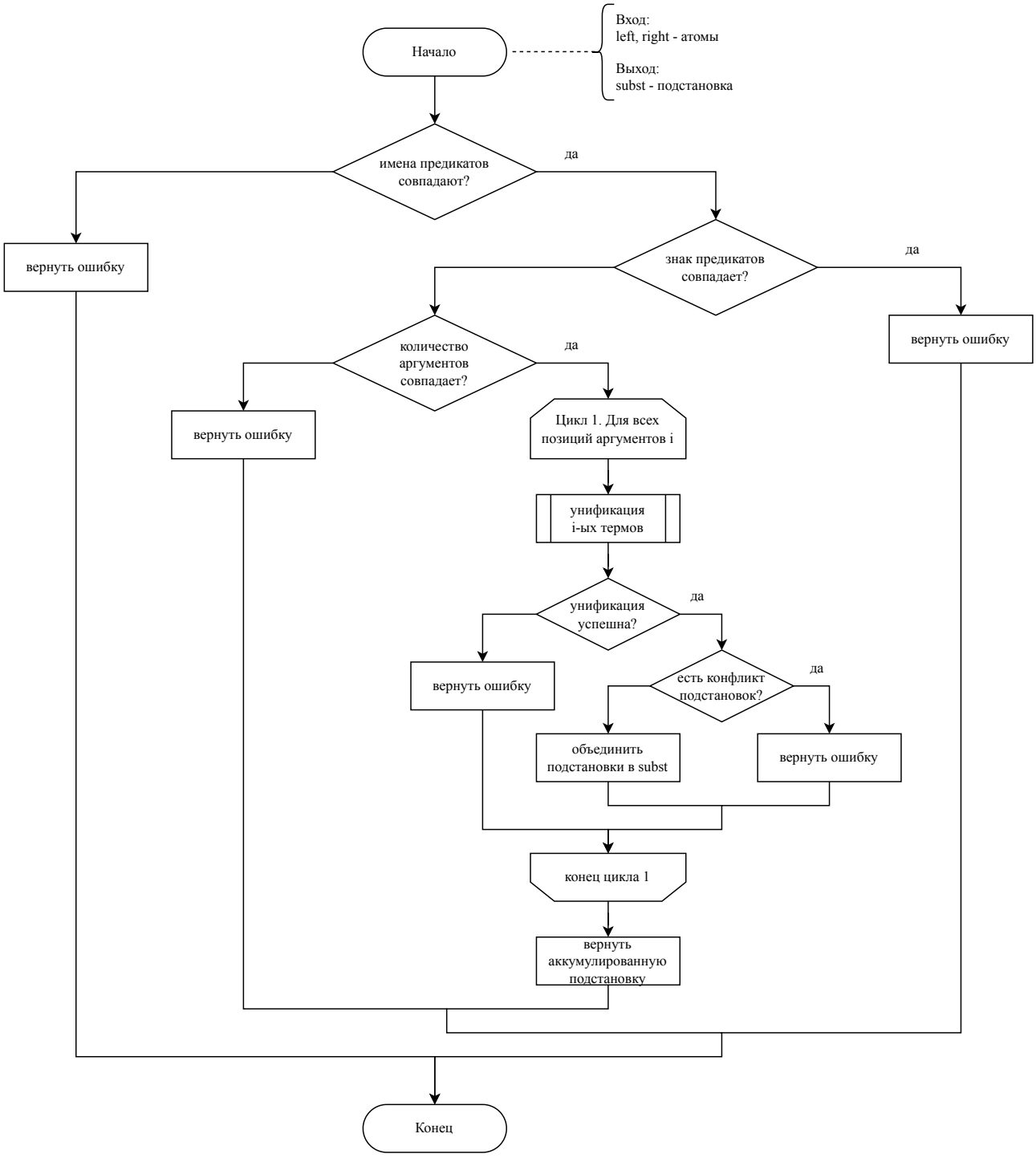

Рисунок 2.1 – Схема алгоритма унификации двух атомов

Определение конфликта подстановок. Так как в процессе унификации распространение значения переменных не используется, необходимо отслеживать непротиворечивость генерируемых подстановок. **Конфликт двух подстановок** определяется выполнением хотя бы одного условия:

1. имеются две переменные в этих подстановках, имеющие одинаковые имена, но связанные с разными (неунифицируемыми) значениями;

2. существует список связанных, но не означенных переменных в одной подстановке, для которой найдутся две переменные, связанные с разными значениями в другой подстановке.

Назовем множество имен A **непротиворечивым на подстановке** B, если все имена из A либо не связаны со значениями в B, либо связаны с одним и тем же значением в B.

Объединение подстановок. Объединение подстановок (назовем их A и B) происходит следующим образом:

1. если подстановка A пуста, то вернуть B;

2. если подстановка B пуста, то вернуть A;

3. создать новую подстановку $C := A$;

4. для всех пар [имя переменной v, значение t] из списка связанных значений подстановки B:

   (a) если имя переменной v связано со значением u в подстановке C и $t \neq u$:

      i. запустить процесс унификации для t и u;

      ii. если унификация неуспешна — вернуть ошибку;

      iii. иначе применить полученную подстановку к значению u;

      iv. связать имя переменной v с новым значением u;

   (b) иначе связать имя переменной v со значением t в C;

5. для всех множеств имен связанных переменных S в подстановке B:

   (a) для всех множеств имен связанных переменных T в подстановке C:

      i. если $S \cap T \neq \emptyset$, то переход на следующую итерацию цикла;

      ii. расширить множество имен переменных $T \leftarrow S \cup T$;

      iii. проверить **непротиворечивость множества имен** T в C;

      iv. если полученное множество противоречиво — вернуть ошибку;

v. иначе — завершить цикл;

(b) если на предыдущем цикле не было расширено ни одно множество, то проверить на непротиворечивость множество имен S в C;

(c) если множество противоречиво – вернуть ошибку;

(d) иначе — добавить множество имен связанных переменных S в C;

6. для для всех пар [имя переменной v, значение t] из списка связанных значений подстановки C:

   (a) **разрешить рекурсию** для значения t в C;

7. вернуть подстановку C.

Алгоритм разрешения рекурсии подставляет значения переменных в функциональные символы, которые были связаны с другими переменными во время слияния подстановок. Рассмотрим пример слияния неконфликтующих подстановок $\{x = f(y)\}$ и $\{y = g(x)\}$. Схема данного процесса изображена на рисунке 2.2. Цель данного алгоритма — избавиться от ссылок на переменные связанные со значениями в данной подстановке из функциональных символов присутствующих в этой подстановке.

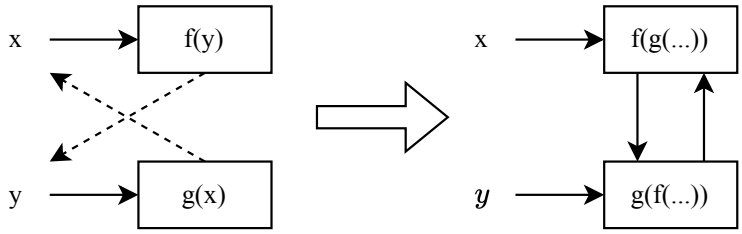

Рисунок 2.2 – Схема разрешения рекурсии

Шаги алгоритма разрешения рекурсии для значения t подстановки C:

1. если t — не функциональный символ, завершить процедуру;

2. для всех агрументов функционального символа t:

   (a) если аргумент — переменная, то заменить ее результатом применения подстановки C к этой переменной;

   (b) иначе если аргумент — функциональный символ, то **разрешить рекурсию** для этого функционального символа.

2.3 Алгоритм унификации двух термов

Схема алгоритма унификации двух термов представлена на рисунке 2.3.

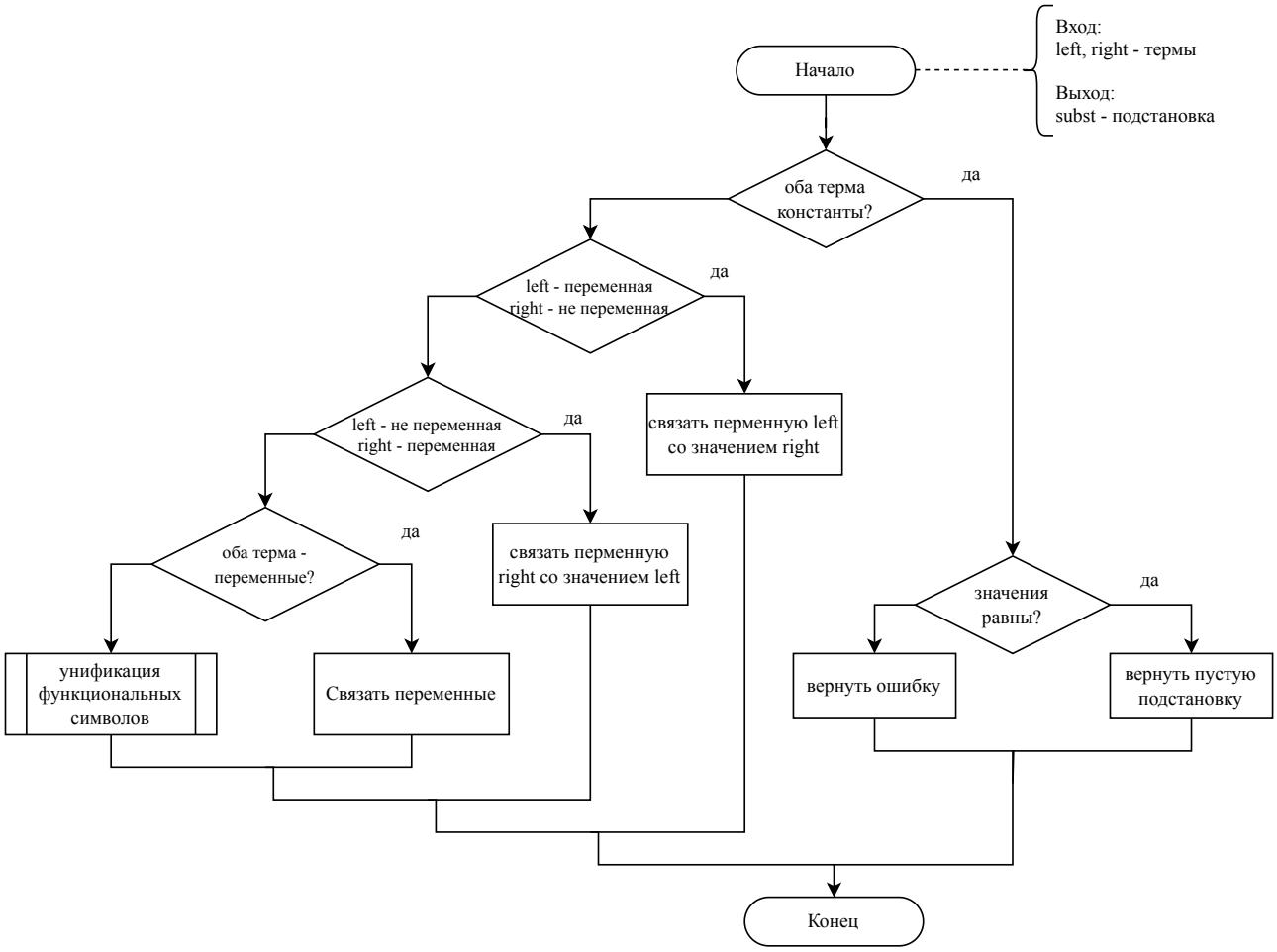

Рисунок 2.3 – Схема алгоритма унификации двух термов

Унификация функциональных символов производится рекурсивно для аргументов функции после проверки соответствия имен функций и количества аргументов согласно представленному алгоритму. После чего осуществляется композиция подстановок. В случае возникновения конфликтов, определяется, что исходные функциональные символы не унифицируемы.

Так как составленная грамматика выражений логики предикатов первого порядка не допускает записи рекурсивных функциональных символов, то и унифицировать их не нужно.

2.4 Алгоритм поска резольвенты для двух дизъюнктов

Алгоритм поиска резольвенты для двух дизъюнктов заключается в простом переборе всех возможных пар атомов из исходных дизъюнктов и попытке их унификации. Если унификация успешна, то конструируется новый дизъюнкт, состоящий из всех атомов исходных дизъюнктов за исключением унифицированной пары. После составления дизъюнкта к нему применяется полученная на этапе унификации подстановка и результат возвращается из процедуры резолюции двух дизъюнктов.

Алгоритм применения подстановки к дизъюнкту заключаеся в последовательном применении подстановки ко всем атомам этого дизъюнкта.

Алгоритм применения подстановки к атому заключается в последовательном применении подстановки ко всем аргументам этого атома (если они есть).

Алгоритм применения подстановки к терму на входе и на выходе имеет терм. Шаги алгоритма:

1. если терм — константа, то вернуть этот же терм;
2. если терм — переменная:
   (a) если переменная есть в списке означенных переменных, то вернуть значение, соответствующее этой переменной;
   (b) иначе для всех списков связанных переменных проверить, если имя терма принадлежит текущему списку связанных переменных, то вернуть первое имя из этого списка;
   (c) иначе вернуть этот же терм;
3. иначе если у терма нет аргументов, то вернуть этот терм;
4. иначе применить подстановку ко всем аргументам и вернуть новый функциональный символ с полученным списком аргументов.

2.5 Алгоритм резолюции

Алгоритм резолюции состоит из полного последовательного перебора всех возможных пар дизъюнктов и поиска для них резольвенты. В случае

успешного нахождения резольвента добавляется в конец списка дизъюнктов и процесс продолжается. Условием останова является получение пустой резольвенты, либо полный перебор с учетом вновь добавленных резольвент, либо достижение максимального количества итераций.

2.6 Диаграмма классов

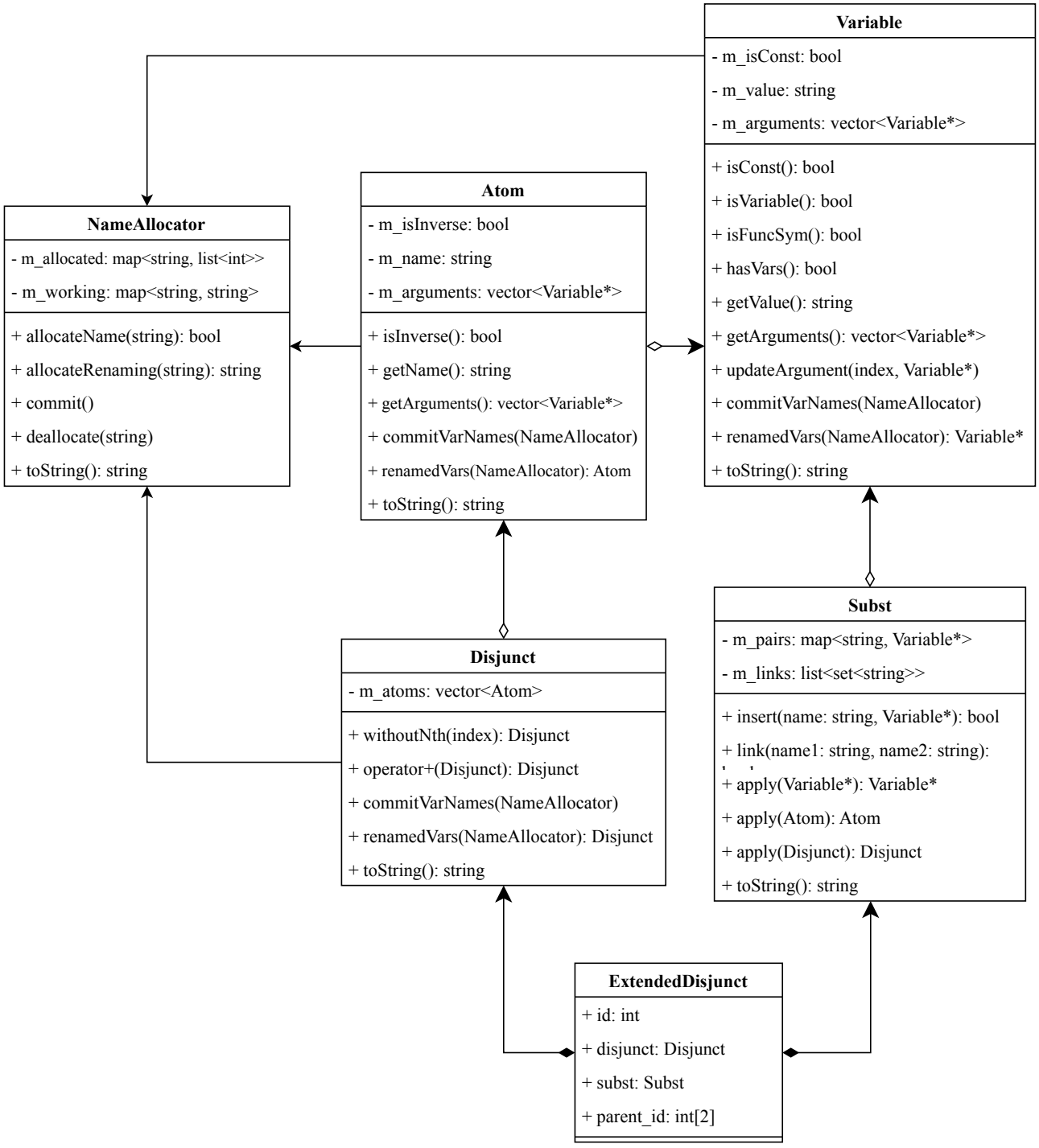

Рисунок 2.4 – Диаграмма классов программы

2.7 Сборка и запуск ПО

Для сборки программы используется `CMake`. Стандартная последовательность команд для сборки и запуска программы:

```
mkdir build && cd build && cmake ..
make app && ./app # сборка и запуск программы
make unittests && ./unittests # опциональный запуск юнит-тестов
```

Альтернативно, можно собрать и запустить программу с помощью `makefile` скрипта в корневой папке используя цель «`app`».

Основная программа работает в цикле REPL (read-evaluate-print loop). Сначала нужно выбрать режим работы. Допустимые значения — `u` для унификации единичных атомов и `r` для запуска алгоритма резолюции. В режиме резолюции первой строкой необходимо ввести список аксиом в формате выражений описанных EBNF грамматикой отделяя выражения символом «`;`». Во второй строке необходимо ввести одно выражение, представляющее цель (не ее отрицание). Пример работы программы:

```
repl for resolution method:
select mode (u - unify, r - resolve): u
enter 'end' to exit repl
enter first atom: P(x, f(x))
enter second atom: P(g(y), y)
unified successfully, subst: {x=g(f(...)), y=f(g(...))}
enter first atom: end
select mode (u - unify, r - resolve): r
enter 'end' to exit repl
enter axioms (colon-separated): \forall(x) (P(x) -> Q(x)); P(A)
enter conclusion: Q(A)
1) ~Q(A)
2) ~P(x1) + Q(x1)
3) P(A)
4) ~P(A) (1 + 2) {x1=A}
5)  (3 + 4) {}
resolved: yes
```

2.8 Тестирование ПО

2.8.1 Тестирование унификации двух атомов

В таблице 2.1 представлены тестовые данные для проверки корректности работы алгоритма унификации двух атомов.

Таблица 2.1 – Тестовые данные для проверки реализации алгоритма унификации

Исходные атомы	Подстановка	Описание
$R(x)$ $R(A)$	$\{x = A\}$	Успешная постановка
$P(x, f(x, B, g(y)), g(C))$ $P(A, f(A, B, z), g(y))$	$\{x = A,\ y = C,$ $z = g(C)\}$	Обновление значений переменных
$Q(S, x, F, G, y, H)$ $Q(y, A, F, G, x, H)$	$\emptyset$	Конфликт связанных переменных
$P(x, f(g(x)))$ $P(y, y)$	$\{x = y = f(g(...))\}$	Рекурсивная подстановка
$H(x, g(z), z, o(x))$ $H(f(y), y, t(x), o(x))$	$\{x = f(g(t(...))),$ $y = g(t(f(...))),$ $z = t(f(g(...)))\}$	Косвенная рекурсия в подстановке

2.8.2 Тестирование алгоритма нахождения резольвенты для двух дизъюнктов

Таблица 2.2 – Тестовые данные для проверки алгоритма поиска резольвенты

Пара дизъюнктов	Резольвента	подстановка
$A, \quad \neg A + B$	B	$\emptyset$
$\neg A + P(f_0), \quad Q(x) + \neg P(x)$	$\neg A + Q(f_0)$	$\{x = f_0\}$
$\neg P(x, y) + \neg P(y, z) + P(x, z)$ $\neg P(3, 5)$	$\neg P(3, y) + \neg P(y, 5)$	$\{x = 3,$ $z = 5\}$

2.8.3 Тестирование алгоритма резолюции

Сценарии тестирования алгоритма резолюции представлены в таблице 2.3.

Таблица 2.3 – Сценарии тестирования алгоритма резолюции

Номер	Исходная задача
1	Аксиомы: $\forall x \forall y \forall z(Len(x,y) \to Len(c(z,x),s(y)))$ $Len(Nil,0)$ Цель: $\exists x Len(x,s(s(0)))$
	Цепочка вывода Аксиомы: $\neg Len(x,y) + Len(c(z,x),s(y)), Len(Nil,0)$ Отрицание цели: $\neg Len(x,s(s(0)))$ 1) $\neg Len(x,s(s(0)))$ 2) $\neg Len(x_1,y_1) + Len(c(z_1,x_1),s(y_1))$ 3) $Len(Nil,0)$ 4) $\neg Len(x_2,s(0))$ $(1+2)$ $\{x = c(z_1,x_1), y_1 = s(0)\}$ 5) $Len(c(z_2,Nil),s(0))$ $(2+3)$ $\{x_1 = Nil, y_1 = 0\}$ 6) $\square$ $(4+5)$ $\{x_2 = c(z_2,Nil)\}$
2	Аксиомы: $\forall x(S(x) + M(x))$ $\neg\exists x(M(x)\&L(x,Lena))$ $\forall x(S(x) \to L(x,Snow))$ $\forall y(L(Lena,y) = \neg L(Petya,y))$ $L(Petya,Rain)$ $L(Petya,Snow)$ Цель: $\exists x(M(x)\&\neg S(x))$
	Цепочка вывода Аксиомы: $S(x) + M(x), \neg M(x_2) + \neg L(x_2,Lena),$ $\neg S(x_2) + L(x_2,Snow), \neg L(Lena,y) + \neg L(Petya,y),$ $L(Petya,y) + L(Lena,y), L(Petya,Rain), L(Petya,Snow)$ Отрицание цели: $\neg M(x) + S(x)$ 1) $\neg M(x) + S(x)$ 2) $S(x_1) + M(x_1)$ 3) $\neg S(x_4) + L(x_4,Snow)$ 4) $\neg L(Lena,y_1) + \neg L(Petya,y_1)$ 5) $L(Petya,Snow)$ 6) $\neg M(x_6) + L(x_6,Snow)$ $(1+3)$ $\{x = x_4\}$ 7) $M(x_7) + L(x_7,Snow)$ $(2+3)$ $\{x_1 = x_4\}$ 8) $\neg L(Lena,Snow)$ $(4+5)$ $\{y_1 = Snow\}$ 9) $\neg M(Lena)$ $(6+8)$ $\{x_6 = Lena\}$ 10)$M(Lena)$ $(7+8)$ $\{x_7 = Lena\}$ 11)$\square$ $(9+10)$ $\{\}$

Выводы

В ходе лабораторной работы была формализована грамматика выражений алгебры логики предикатов первого порядка, разработана программа для автоматического разбора и доказательства произвольных выражений на языке алгебры логики предикатов первого порядка.