Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к выпускной квалификационной работе

«Автоматическое доказательство теорем на основе Conflict Resolution»

Автор: Итегулов Данияр Саматович _____
Направление подготовки (специальность): 01.03.02 Прикладная математика и информатика
Квалификация: Бакалавр
Руководитель: Ульянцев В.И., канд. техн. наук _____

К защите допустить
Зав. кафедрой Васильев В.Н., докт. техн. наук, проф. _____

«___» _____ 20___ г.

Санкт-Петербург, 2017 г.

Студент Итегулов Д.С. **Группа** М3439 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования

Направленность (профиль), специализация Математические модели и алгоритмы в разработке программного обеспечения

Квалификационная работа выполнена с оценкой _____

Дата защиты «___» _____ 20___ г.

Секретарь ГЭК *Павлова О.Н.* Принято: «___» _____ 20___ г.

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

УТВЕРЖДАЮ
Зав. каф. компьютерных технологий
докт. техн. наук, проф.
_____ Васильев В.Н.
«___» _____ 20___ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студент Итегулов Д.С. **Группа** M3439 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования **Руководитель** Ульянцев В.И., канд. техн. наук, доц. каф. КТ

1 Наименование темы: Автоматическое доказательство теорем на основе Conflict Resolution

Направление подготовки (специальность): 01.03.02 Прикладная математика и информатика

Направленность (профиль): Математические модели и алгоритмы в разработке программного обеспечения

Квалификация: Бакалавр

2 Срок сдачи студентом законченной работы: «___» _____ 20___ г.

3 Техническое задание и исходные данные к работе.

Требуется разработать алгоритм на основе исчисления *Conflict Resolution*, который бы позволил обобщить *CDCL* на теорию первого порядка. Реализация должна быть полной относительно опровержения и уметь эффективно решать задачи в конъюнктивной нормальной форме из библиотеки *TPTP*.

4 Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)

а) Обзор предметной области
б) Разработка алгоритмов на основе исчисления *Conflict Resolution*
в) Реализация предложенных алгоритмов, проведение экспериментов и сравнение результатов с существующими решениями

5 Перечень графического материала (с указанием обязательного материала)

Не предусмотрено

6 Исходные материалы и пособия

а) *Slaney J.*, *Woltzenlogel Paleo B.* Conflict Resolution: a First-Order Resolution Calculus with Decision Literals and Conflict-Driven Clause Learning // Journal of Automated Reasoning. 2017. P. 1–24

б) Исходный код программного средства Skeptik

7 Календарный план

№№ пп.	Наименование этапов выпускной квалификационной работы	Срок выполнения этапов работы	Отметка о выполнении, подпись руков.
1	Ознакомление с предметной областью	01.10.2016	
2	Ознакомление с кодовой базой программного средства *Skeptik*	01.11.2016	
3	Ознакомление с исчислением *Conflict Resolution*	15.11.2016	
4	Построение алгоритмов, удовлетворяющих требованиям	01.01.2016	
5	Реализация предложенных алгоритмов	01.03.2017	
6	Экспериментальное исследование алгоритмов и сравнение с другими системами автоматических доказательств	01.04.2017	
7	Написание пояснительной записки	01.06.2017	

8 Дата выдачи задания: «01» сентября 2016 г.

Руководитель _____

Задание принял к исполнению _____ «01» сентября 2016 г.

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Студент: Итегулов Данияр Саматович

Наименование темы работы: Автоматическое доказательство теорем на основе Conflict Resolution

Наименование организации, где выполнена работа: Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования: Разработать алгоритм автоматического доказательства теорем, обобщающий пропозициональный метод CDCL на теорию первого порядка с помощью исчисления Conflict Resolution

2 Задачи, решаемые в работе:

а) описание сложностей обобщения CDCL на теорию первого порядка;

б) разработка алгоритмов и доказательство их полноты относительно опровержений;

в) реализация алгоритмов на языке программирования Scala.

3 Число источников, использованных при составлении обзора: 16

4 Полное число источников, использованных в работе: 20

5 В том числе источников по годам

Отечественных			Иностранных		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
0	0	0	9	3	8

6 Использование информационных ресурсов Internet: нет

7 Использование современных пакетов компьютерных программ и технологий: Был использован язык программирования Scala, интегрированная среда разработки IntelliJ Idea, система автоматического тестирования scalatest и система компьютерной верстки LaTeX.

8 Краткая характеристика полученных результатов: Разработаны алгоритмы автоматического доказательства теорем, которые на наборе задач сопоставимы по эффективности с существующими системами.

9 Гранты, полученные при выполнении работы: Работа была частично проспонсирована программой Google Summer of Code 2016.

10 Наличие публикаций и выступлений на конференциях по теме работы: Работа была принята на международную конференцию CADE-26:
Itegulov D., *Paleo B. W.*, *Slaney J.* Scavenger 0.1: A Theorem Prover Based on Conflict Resolution // Proceedings of the 26th International Conference on Automated Deduction (CADE). 2017. Forthcoming

Выпусник: Итегулов Д.С. _____

Руководитель: Ульянцев В.И. _____

«___» _____ 20___ г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Автоматическое доказательство теорем – один из разделов математической логики, находящий свое применение в таких сферах как верификация интегральных схем, верификация программного обеспечения и теория искусственного интеллекта. К примеру, Microsoft Research используют систему автоматического доказательства теорем Z3 для верификации программного кода семейства операционных систем Windows и различных сетевых протоколов собственной разработки.

Основной целью данной работы является разработка эффективного метода поиска доказательств в теории первого порядка, обладающей высокой выразительной силой для описания большого числа моделей из различных сфер математики (линейная алгебра, математический анализ, биоинформатика, верификация программного обеспечения и другое).

Существующие подходы к автоматическому доказательству теорем используют методы «насыщения» модели путем генерации новых утверждений на основе уже существующих (и, в основном, используют резолюционное исчисление). Таким образом главным недостатком существующих методов является слишком «широкий» поиск доказательства. С другой стороны, в классической задаче SAT существуют различные способы вывода новых дизъюнктов, основанные на выводе новой информации из предположений и конфликтов, которые они влекут. Данная работа пытается обобщить такие способы на теорию первого порядка: будут рассмотрены алгоритмы PDCR, EPCR и TDCR, которые обобщают пропозициональный алгоритм CDCL с помощью исчисления Conflict Resolution.

Данная тема интересна в первую очередь из-за того, что CDCL очень хорошо себя показывает на реальных задачах из сферы верификации программного обеспечения. Обобщив его на теорию первого порядка, можно надеяться на получения алгоритма, который будет также хорошо работать на задачах из этой сферы.

Модель многих практических задач выглядит значительно проще, если описывать ее в теории первого порядка: не нужно никак описывать предметное множество и даже функции, используемые в аксиомах.

Работа была принята на конференцию CADE-26, проходящую 8-го августа 2017-го года в Гетеборге, Швеция.

В главе 1 будут рассмотрены классическая задача SAT, алгоритмы DPLL, CDCL, постановка решаемой задачи и исчисление Conflict Resolution.

В главе 2 будут показаны основные проблемы при обобщении CDCL на теорию первого порядка и предоставлены алгоритмы решения вышеперечисленных проблем, а также доказательства их полноты относительно опровержения.

В главе 3 будет кратко описана реализация системы и предоставлены сравнения с существующими подобными средствами на конкретных примерах и сделаны выводы по классу задач, которые он умеет решать лучше и хуже.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ

1.1. Задача SAT и алгоритмы DPLL, CDCL

1.1.1. Постановка задачи SAT

Определение 1. *Пропозициональная формула* — индуктивная структура, определяемая следующим образом:

а) Всякая пропозициональная переменная есть формула (также называемая атомом)

б) Если A — формула, то $\neg A$ также является пропозициональной формулой

в) Если A и B — формулы, то $A \vee B$, $A \wedge B$, $A \to B$ также являются пропозицианальными формулами

Определение 2. Пусть $V = \{P_1, P_2, \ldots\}$ – множество всех пропозициональных переменных. *Оценкой* называется произвольное отображение $g\colon V \to \{0, 1\}$. Индукцией по построению формулы A зададим значение $g(A)$ формулы A:

а) Для любого атома A значение $g(A)$ задано оценкой g

б) $g(\neg A) = \neg g(A)$

в) $g(A \vee B) = g(A) \vee g(B)$

г) $g(A \wedge B) = g(A) \wedge g(B)$

д) $g(A \to B) = g(A) \to g(B)$

Определение 3. *Задача удовлетворимости пропозициональной формулы* (или *задача SAT*) – задача поиска такой оценки g для некоторой пропозициональной формулы A, что $g(A) = 1$.

Определение 4. Пропозициональная формула находится в *конъюнктивной нормальной форме* (*CNF*), если она является конъюнкцией дизъюнктов.

Пример 5.

— $\neg A \wedge (B \vee C)$

— $(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$

Теорема 6. Любая пропозициональная формула может быть представлена в эквивалентной конъюнктивной нормальной форме.

Таким образом мы будем решать изоморфную задачу *CNFSAT*, которая ограничивается рассмотрением формул в виде *CNF*. Это классическая *NP*-полная задача, но в некоторых реальных случаях ее можно

решить не перебирая все возможные оценки. Следующие несколько параграфов посвящены известным алгоритмам, которые эффективно решают *CNFSAT* на реальных задачах.

1.1.2. Алгоритм DPLL

DPLL – один из алгоритмов решения задачи *CNFSAT*, впервые представленный в 1962-ом году [3], который использует правило резолюций.

Определение 7. Пусть P – это пропозициональная переменная и Δ – это пропозициональная формула в конъюнктивной нормальной форме, содержащая дизъюнкты C_i, C_j, где $P \in C_i, \neg P \in C_j$. Тогда *правило резолюций* позволяет вывести новый дизъюнкт $(C_i - \{P\}) \cup (C_j - \{\neg P\})$, который также называется резолвентой C_i и C_j.

Пример 8. Резолвентой $\{A, B, \neg C\}$ и $\{\neg B, D\}$ является $\{A, \neg C, D\}$.

Правило резолюций корректно, но не обладает полнотой в том смысле, что оно не позволяет вывести все возможные верные дизъюнкты. Тем не менее оно обладает другим важным свойством:

Определение 9. Если исходная формула является противоречивой, то с помощью правила резолюций всегда можно вывести пустой дизъюнкт ($\perp$). Данное свойство логической системы называется *полнотой относительно опровержения*.

Определение 10. Пусть Δ – это пропозициональная формула в конъюнктивной нормальной форме, тогда *ветвлением* Δ по литералу L называется следующая операция:
$\Delta|L = \{\alpha - \{\neg L\} | \alpha \in \Delta, L \notin \alpha\}$

Пример 11. Пусть $\Delta = \{\{A, B, \neg C\}, \{\neg A, D\}, \{B, C, D\}\}$. Тогда:
- $\Delta|C = \{\{A, B\}, \{\neg A, D\}\}$
- $\Delta|\neg C = \{\{\neg A, D\}, \{B, D\}\}$

Правило ветвления также можно представить как разбор 3 случаев дизъюнктов при ветвлении по переменной L:

а) Дизъюнкт содержит переменную L. Так как мы заменили L на *истину*, то в $\Delta|L$ данный дизъюнкт участвовать уже не будет (он уже удовлетворен).

б) Дизъюнкт содержит переменную $\neg L$. Так как мы заменили L на *истину*, то в $\Delta|L$ данный дизъюнкт примет участие, но уже без $\neg L$ (этот литерал уже никак не может быть удовлетворен).

в) Дизъюнкт не содержит ни L, ни $\neg L$. Тогда этот дизъюнкт примет участие в $\Delta|L$ без изменений.

Определение 12. Дизъюнкт называется *единичным*, если в нем содержится ровно один литерал.

Листинг 1 – DPLL

```
function DPLL(Δ : CNF)
    (I, Γ) = UnitResolution(Δ)
    if Γ = {} then
        return {}
    else if True then
        return ⊥
    else
        Choose a literal L in Γ
        if P = DPLL(Γ|L) ≠ ⊥ then
            return P ∪ I ∪ {L}
        else if P = DPLL(Γ|¬L) ≠ ⊥ then
            return P ∪ I ∪ {¬L}
        else
            return ⊥
        end if
    end if
end function
```

Заметим, что если дизъюнкт является единичным, то удовлетворить его можно только оценив данную переменную в необходимое нам значение (т.е. если дизъюнкт содержит $\neg A$, то A нужно оценить в ложь и наоборот). Это простое соображение на практике сильно сужает пространство поиска оценки для удовлетворения формулы.

Определение 13. Описанная выше операция называется *единичной резолюцией*.

Алгоритм $DPLL$ описан в листинге 1 и заключается в следующем:

а) Применяем единичную резолюцию

б) Выбираем литерал из оставшихся дизъюнктов

в) Делаем ветвление по выбранному литералу и если оно ведет к противоречию, то делаем ветвление по отрицанию выбранного литерала

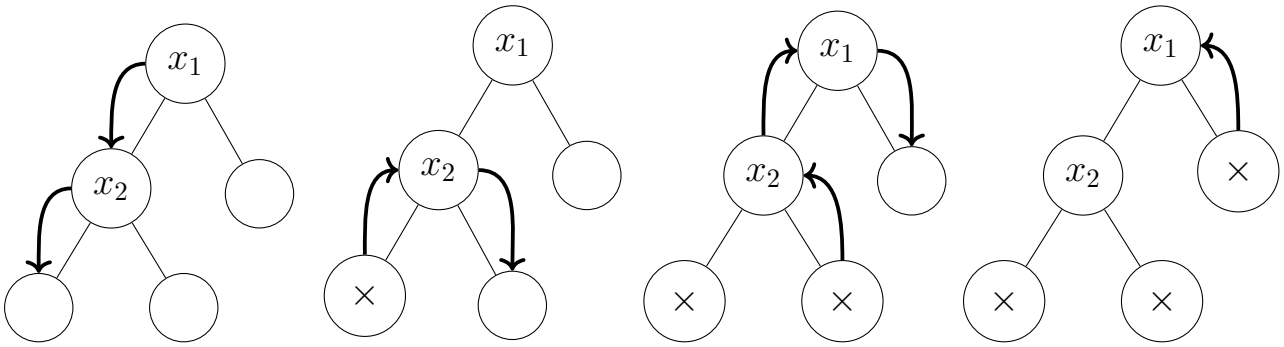

Рисунок 1 – Шаги *DPLL*

г) Повторяем, пока не получим оценку в которой данная формула верна или пока не проверим все оценки формулы (тогда формула противоречива).

Пример 14. Рассмотрим пример работы алгоритма *DPLL* на следующей формуле (описанного в виде множества дизъюнктов):

$$\begin{array}{lll} \neg x_1 \vee x_3 \vee x_4, & \neg x_2 \vee x_6 \vee x_4, & \neg x_2 \vee \neg x_6 \vee \neg x_3, \\ \neg x_4 \vee \neg x_2, & x_2 \vee \neg x_3 \vee \neg x_1, & x_2 \vee x_6 \vee x_3, \\ x_2 \vee \neg x_6 \vee \neg x_4, & x_1 \vee x_5, & x_1 \vee x_6, \\ \neg x_6 \vee x_3 \vee \neg x_5, & x_1 \vee \neg x_3 \vee \neg x_5 & \end{array}$$

После оценивания x_1 и x_2 в истину (с помощью операции ветвления как показано на рисунке 1 (а)) изначальная формула преобразуется в:

$$\{x_3 \vee x_4, x_6 \vee x_4, \neg x_6 \vee \neg x_3, \neg x_4\}$$

Правило единичной резолюции из $\neg x_4$ выведет x_3 и x_6. Но это противоречит дизъюнкту $\neg x_6 \vee \neg x_3$. На рисунке 1 (б) показано как после получения данного противоречия происходит процесс отката до состояния, где x_2 еще не было оценено в истину и x_2 оценивается в ложь. Тогда формула преобразовывается в:

$$\{x_3 \vee x_4, \neg x_3, x_6 \vee x_3, \neg x_6 \vee \neg x_4\}$$

Из $\neg x_3$ по правилу единичной резолюции выводятся x_4 и x_6, которые противоречат дизъюнкту $\neg x_6 \vee \neg x_4$. На рисунке 1 (в) показано как было получено очередное противоречие и произошел откат до изначального состояния (т.е. никаких оценок нет), после чего x_1 оценили в истину. Получилась формула:

$$\begin{array}{rrr} \neg x_2 \vee x_6 \vee x_4, & \neg x_2 \vee \neg x_6 \vee \neg x_3, & \neg x_4 \vee \neg x_2, \\ x_2 \vee x_6 \vee x_3, & x_2 \vee \neg x_6 \vee \neg x_4, & x_5, \\ x_6, & \neg x_6 \vee x_3 \vee \neg x_5, & \neg x_3 \vee \neg x_5 \end{array}$$

После вывода по правилу единичной резолюции $\neg x_3$ из x_5, получается противоречие с дизъюнктом $\neg x_6 \vee x_3 \vee \neg x_5$. Рисунок 1 (г) показывает как было достигнуто противоречие на последней вершине и вся формула объявляется неудовлетворимой.

1.1.3. Алгоритм CDCL

Одной из основных причин широко распространенного использования задачи *SAT* является то, что алгоритм *CDCL* [4] и системы основанные на нем так эффективны на практике. *CDCL* является расширением алгоритма *DPLL*, которое выводит новую информацию из достигнутых конфликтов.

Для того чтобы формально описать процедуру *CDCL*, нам понадобится следующие определения:

Определение 15. Импликационный граф – ориентированный граф $G(V, E)$, каждая вершина которого $v \in V$ представляет собой значение истинности булевского литерала, а каждое ребро $e \in E = (u, v)$ обозначает импликацию «если литерал u верен, то литерал v также верен».

Определение 16.

— Если литерал ℓ был предположен, то номер данного предположения является *уровнем* ℓ

— Если литерал ℓ был выведен в процессе распространения литералов после того как было сделано предположение под номером k, то *уровнем* ℓ является k.

Уровень литерала ℓ также обозначается как $\delta(\ell)$

Определение 17. *Оценкой переменных для получения литерала* ℓ называется такая оценка переменных, которая влечет ℓ с помощью процесса распространения литералов. Также обозначается как $A(\ell)$.

Определение 18. *Конфликтной оценкой переменных* называется оценка переменных для получения $\bot$.

Определим две вспомогательные функции, действующие на литералах из импликационного графа:

$$\Lambda(\ell) = \{(y = \nu(y)) \in A(\ell) | \delta(y) < \delta(\ell)\}$$
$$\Sigma(\ell) = A(x) \setminus Lambda(\ell)$$

Предположим мы запустили процедуру *DPLL* и пришли к некоторому конфликту $\bot$. Посмотрим на импликационный граф в этот момент времени. Найдем конфликтную оценку для $\bot$ с помощью функции $CauseOf(\ell)$:

$$CauseOf(\ell) = \begin{cases} \ell = \nu(\ell), & \text{если } A(\ell) = \emptyset \\ \Lambda(\ell) \cup (\bigcup\limits_{(\ell' = \nu(\ell')) \in \Sigma(\ell)} CauseOf(\ell')), & \text{иначе} \end{cases}$$

Тогда конфликтной оценкой для $\bot$ будет являться $A_\bot = CauseOf(\bot)$. А отрицания всех литералов из $A_\bot$, соединенных через дизъюнкцию, образуют дизъюнкт, который также должен быть верен в изначальной постановке задачи. Таким образом мы научились получать новый дизъюнкт из конфликта. После вывода этого нового дизъюнкта, также необходимо сделать откат до максимального из уровней литералов из $A_\bot$. В частности, для самого первого конфликта текущий уровень не изменится, а последнее предположение просто сменит полярность. Однако последующие конфликты могут произвести откат к уровню, отличному от текущего.

Листинг 2 показывает, как реализовывается типичная система решения задачи *SAT* на основе алгоритма *CDCL*. Основным отличием данного листинга от листинга 1 является вызов функции ConflictAnalysis при получении конфликта и вызов функции Backtrack при необходимости. Функция Backtrack (потенциально нехронологически) откатывает состояние Δ до нужного состояния и добавляет туда новый *CDCL* дизъюнкт.

Листинг 2 – CDCL

```
function CDCL(Δ : CNF, ν : Assignment)
    if UnitPropagation(Δ, ν) == ⊥ then
        return ⊥
    end if
    while ¬ AllVariablesAssigned(Δ, ν) do
        Choose the branching literal L in Γ
        dl ← dl + 1
        ν ← ν ∪ {L}
        if UnitPropagation(Δ, ν) == ⊥ then
            β ← ConflictAnalysis(Δ, ν)
            if β is Nothing then
                return ⊥
            else
                Backtrack(Δ, ν, β)
                dl ← β
            end if
        end if
    end while
end function
```

Помимо прочего, в листинге 2 также используются функции:

— UnitPropagation – последовательно применяет единичную резолюцию. Если в какой-то момент был получен пустой дизъюнкт, то возвращает $\bot$.

— ConflictAnalysis – ищет самый последний конфликт в Δ и добавляет туда новый дизъюнкт. Также возвращает номер уровня к которому необходимо откатиться. Есть несколько подходов к реализации данного метода, описание которых выходит за рамки данной работы.

— Backtrack – откатывается к уровню, подсчитанному функцией ConflictAnalysis

— AllVariablesAssigned – проверяет, была ли выставлена оценка всем переменным формулы. На самом деле, данный факт в эффективно реализованных системах проверяется лениво.

Пример 19. В качестве примера работы *CDCL* рассмотрим получение конфликта и вывод из него дизъюнкта на формуле:

$$\begin{array}{ccc} \neg x_1 \vee \neg x_4 \vee x_5, (\omega_1) & \neg x_4 \vee x_6, (\omega_2) & \neg x_5 \vee \neg x_6 \vee x_7, (\omega_3) \\ \neg x_7 \vee x_8, (\omega_4) & \neg x_2 \vee \neg x_7 \vee x_9, (\omega_5) & \neg x_8 \vee \neg x_9, (\omega_6) \\ x_4 \vee x_7 (\omega_7) & \neg x_9 \vee \neg x_7 (\omega_8) & x_3 \vee x_7 \vee \neg x_4 (\omega_9) \end{array}$$

Сделаем 4 предположения x_1, x_2, x_3 и x_4.

В данной нотации вершина $x_1@1$ обозначает, что x_1 была получена на первом шаге алгоритма. Импликационный граф после распространения литералов выглядит, как показано на рисунке 2. Как мы видим, был достигнут конфликт и, следовательно, на данном этапе алгоритма нужно вывести новый дизъюнкт на основе информации в данном импликационном графе. Конфликтной оценкой $A_\perp$ здесь является $\{x_1 = 1@1, x_2 = 1@2, x_3 = 1@3, x_4 = 1@4\}$. Получается, что новым дизъюнктом будет $\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4$ (далее обозначается как ω_{10}). Как говорилось выше, после первого конфликта текущий уровень не меняется. Убирается последнее предположение x_4, но из нового выученного дизъюнкта выводится $\neg x_4$.

После очередного запуска процесса распространения литералов, получится импликационный граф, показанный на рисунке 3. В этот раз для вывода конфликта понадобилась конфликтная оценка $A_\perp = \{x_1 = 1@1, x_2 = 1@2\}$. Максимальный уровень в $A_\perp$ равняется двум, поэтому произойдет нехронологический откат к уровню 2.

1.2. Постановка решаемой задачи и ее отличие от SAT

1.2.1. Теория первого порядка

Классическая задача *SAT* широко используется на практике, но иногда необходима более гибкая и сложная теория для описания некоторых математических высказываний. Для таких целей существуют теории высших порядков. Конкретно в данной работе будет рассматриваться теория первого порядка.

Основные отличия теории первого порядка от пропозициональной логики заключаются в использовании квантифицированных перемен-

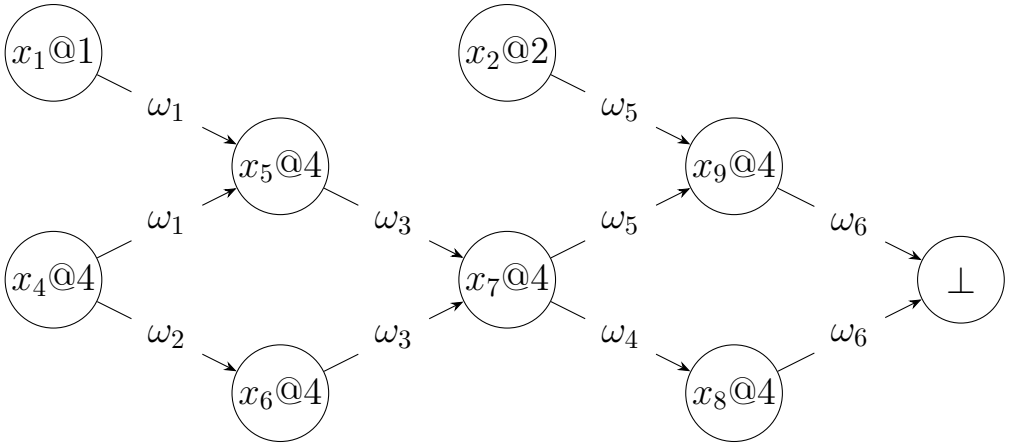

Рисунок 2 – Импликационный граф после распространения ограничений

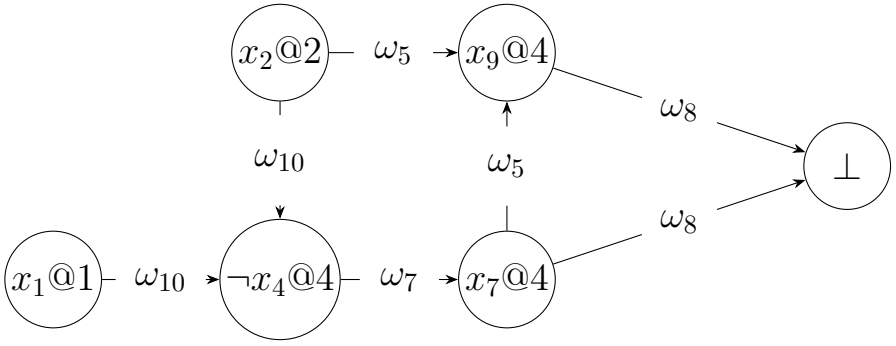

Рисунок 3 – Импликационный граф после вывода ω_{10}

ных над не-булевскими объектами, фиксированных предикатов и функциональных символов.

Более формально:

Определение 20. *Терм* исчисления предикатов – это следующая индуктивная структура:

- предметная переменная – обозначается маленькой буквой начала или конца латинского алфавита, возможно, с индексом или апострофом.
- применение функции (функции мы будем обозначать латинскими буквами середины алфавита: $f, g, h, \ldots$): если $\theta_1 \ldots \theta_n$ – термы и f – *функциональный символ* (то есть символ, обозначающий некоторую функцию), то $f(\theta_1, \ldots \theta_n)$ – тоже терм. Также, частным случаем функций являются константы – нульместные функции. Обычно мы их будем обозначать маленькими буквами c или d, возможно, с индексами.

Определение 21. *Формула* исчисления предикатов – это следующая индуктивная структура:

- если α и β – формулы исчисления предикатов, то $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \to \beta$ – также формула. Связки расставлены в порядке убывания приоритета. Как и в исчислении высказываний, импликация правоассоциативна, остальные операции – левоассоциативны.
- если α – формула и x – предметная переменная, то $\forall x\, \alpha$ и $\exists x\, \alpha$ также являются формулами. Кванторы имеют приоритет, одинаковый с отрицанием, и, как и отрицание, действуют только на ближайшее за ними логическое выражение. Например, формула $\forall x\, P(x,5) \vee P(x,7)$ соответствует формуле $(\forall x\, P(x,5)) \vee P(x,7)$.
- применение предиката (предикаты мы будем обозначать большими латинскими буквами, возможно, с индексами): если $\theta_1 \ldots \theta_n$ – термы, и P – *предикатный символ*, то $P(\theta_1, \ldots \theta_n)$ – формула. В частности, при $n = 0$ предикат становится аналогом предметной переменной из исчисления высказываний.

Формулировать теоремы в таком виде часто бывает довольно удобно и даже существует общепринятый формат *FOF* для записи задач для систем автоматического доказательства теорем. Однако, обычно с фор-

мулами в таком общем виде работать не очень удобно и существует адаптированная к теории первого порядка конъюнктивная нормальная форма формул.

Опишем процедуру конвертации произвольной формулы теории первого порядка в конъюнктивную нормальную форму:

— Приведем формулу к отрицательной нормальной форме
  – Заменим все импликации $P \to Q$ на $\neg P \vee Q$
  – Внесем отрицания внутрь используя *законы Де Моргана*. В частности, $\neg(P \vee Q)$ заменим на $\neg P \wedge \neg Q$, а $\neg(P \wedge Q)$ на $\neg P \vee \neg Q$. Также существуют особенные случаи характерные только для теории первого порядка. Например $\neg(\forall x P(x))$ заменяется на $\exists x \neg P(x)$.
— Стандартизируем имена переменных
  – Для таких формул как $(\forall x P(x)) \vee (\exists x Q(x))$, которые используют одинаковое имя дважды, необходимо поменять имя одной из переменных. Это делается для того, чтобы убрать неоднозначность после элиминации кванторов в последующих шагах.
— Сколемизируем формулу
  – Вынесем кванторы наружу. Например, заменим $P \wedge (\forall x Q(x))$ на $\forall x (P \wedge Q(x))$. Такие преобразования не меняют смысла формулы и, после проведения замен, кванторы могут встречаться только в префиксе конечной формулы.
  – Заменим все вхождения $\forall x_1 \ldots \forall x_n \exists y P(y)$ на $\forall x_1 \ldots \forall x_n P(f(x_1, \ldots, x_n))$, где f – новая n-арная функция (так называемая *функция Сколема*). Данный шаг не сохраняет эквивалентность формулы, но сохраняет ее свойство быть удовлетворимой.
— Опустим все кванторы всеобщности
— Вынесем все конъюнкции наружу с помощью дистрибутивности: заменим $P \vee (Q \wedge R)$ на $(P \vee Q) \wedge (P \vee R)$.

Пример 22. Рассмотрим следующую формулу:

$$\exists x \, \forall y \, \forall z \, (person(x) \wedge ((likes(x, y) \wedge y \neq z) \to \neg likes(x, z)))$$

Удалим все импликации:

$$\exists x \, \forall y \, \forall z \, (person(x) \wedge (\neg(likes(x, y) \wedge y \neq z) \vee \neg likes(x, z)))$$

Внесем все отрицания на самую глубокую позицию:

$$\exists x \, \forall y \, \forall z \, (person(x) \wedge (\neg likes(x, y) \vee y = z \vee \neg likes(x, z)))$$

Все кванторы уже снаружи, поэтому сразу приступаем к сколемизации. Заметим, что перед x нет кванторов всеобщности, поэтому мы заменяем его на константный функциональный символ p (но в общем случае функциональный символ может получиться неконстантным).

$$\forall y \, \forall z \, (person(p) \wedge (\neg likes(p, y) \vee y = z \vee \neg likes(p, z)))$$

Отбросим кванторы всеобщности:

$$person(p) \wedge (\neg likes(p, y) \vee y = z \vee \neg likes(p, z))$$

Итоговая формула находится в конъюнктивной нормальной форме (с неявными кванторами над y и z).

1.2.2. Постановка задачи FO-SAT

Чтобы оценить формулу в теории первого порядка, недостаточно просто дать оценку переменным. Необходимо ввести оценку для предикатных символов: для каждого k-местного предикатного символа P_n^k определить функцию $\nu_{P_n^k} : D^k \to V$ (где D – предметное множество). Аналогично, для каждого k-местного функционального символа f_n^k также нужно определить функцию $\nu_{f_n^k} : D^k \to D$.

Определение 23. Формула в теории первого порядка называется *удовлетворимой* если существует такая оценка (модель), что формула в данной модели оценивается в истину.

Задача перед разрабатываемой системой поставлена следующим образом: необходимо найти оценку, удовлетворяющую данную формулу, или доказать ее противоречивость.

1.2.3. Существующие подходы к решению поставленной задачи

Существует множество различных систем автоматического доказательства теорем в теории первого порядка и большинство используют различные вариации резолюционного исчисления, представленного Робинсоном в 1956-ом году [5]. В нем используется всего одно правило, которое тем не менее обладает свойством полноты относительно опровержения. Это правило – обобщение резолюции на теорию первого порядка.

Определение 24. Пусть P – это терм в теории первого порядка и Δ – это формула в конъюнктивной нормальной форме из теории первого порядка, содержащая дизъюнкты C_i, C_j. Пусть существует $Q \in C_i$, что Q может быть приведено к P путем подстановки некоторых квантифицированных переменных и, аналогично, существует $\neg R \in C_j$, что R может быть приведено к P. Тогда P называется *общим унификатором* Q и R, а *резолвентой* дизъюнктов Q и R называется $C_i \cup C_j \setminus \{Q, \neg R\}$.

Современные системы автоматического доказательства теорем используют технику доказательства через опровержение. Задача для системы записывается следующим образом: список аксиом, предположений и *отрицание* теоремы (т.е. того, что мы хотим доказать). Системы автоматического доказательства теорем на основе резолюций применяют правило резолюций для дизъюнктов снова и снова, пока резолвентой каких-то двух дизъюнктов не окажется *пустой* дизъюнкт ($\perp$). Как только нашлось противоречие, мы можем быть уверены, что теорема верна и восстановить ее доказательство по опровержению ее отрицания.

Пример 25. Рассмотрим пример работы алгоритма доказательства на основе резолюций. Рассмотрим следующую задачу (в каждой строке записан один дизъюнкт):

$$\neg man(X) \vee mortal(X)$$
$$man(s)$$
$$\neg mortal(s)$$

Первый дизъюнкт является аксиомой и говорит о том, что все люди смертны. Второй дизъюнкт – это предположение о том, что Сократ

$$\dfrac{\dfrac{man(s) \qquad \neg man(X) \vee mortal(X)}{mortal(s)}\,\mathbf{r}(\{X \backslash s\}) \qquad \neg mortal(s)}{\bot}\,\mathbf{r}(\{\})$$

Рисунок 4 – Доказательство смертности Сократа

(обозначен за s) является человеком. Последнее же высказывание – это *отрицание* теоремы: утверждение о том, что Сократ смертен. На рисунке 4 приведено доказательство данного факта.

1.3. Conflict Resolution

Оперируя только лишь чистым резолюционным исчислением сложно выразить идеи алгоритма *CDCL* в теории первого порядка. Поэтому было представлено новое исчисление под названием *Conflict Resolution*, главная цель которого – внести правила вывода на уровне исчисления, походящие на техники *DPLL* и *CDCL* в теории первого порядка.

Другие попытки обобщения этих техник включают в себя *Model Evolution* [6–9], *Geometric Resolution* [10], *Non-Redundant Clause Learning* [11] и *Semantically-Guided Goal Sensetive procedure* [12]. Краткий обзор и их сравнение с *Conflict Resolution* можно найти в [1]. Также существует множество систем автоматического доказательства теорем первого и высших порядков, которые используют пропозициональные алгоритмы для некоторых подзадач, не делая попыток обобщить эти техники на более высокие порядки. Среди примеров таких систем можно выделить [13–17].

Правила вывода исчисления *Conflict Resolution* можно увидеть на рисунке 5. Правило *Unit Propagating Resolution* является цепочкой последовательных резолюций с одиночными дизъюнктами, а его выводом является единственный одиночный дизъюнкт. *Предположенные* литералы обозначаются квадратными скобками (например $[\ell]$) и правило *Conflict-Driver Clause Learning* (далее будет использоваться сокращенное название *CDCL*, что вносит некоторую неоднозначность, т.к. алгоритм *CDCL* имеет идентичное название) позволяет выводить новый дизъюнкт состоящий из отрицаний подстановок предположенных литералов, кото-

Unit-Propagating Resolution:

$$\frac{\ell_1 \quad \ldots \quad \ell_n \quad \overline{\ell'_1} \vee \ldots \vee \overline{\ell'_n} \vee \ell}{\ell\,\sigma}\,\mathbf{u}(\sigma)$$

где σ – это унификатор ℓ_k и ℓ'_k для всех $k \in \{1, \ldots, n\}$.

Conflict:

$$\frac{\ell \quad \overline{\ell'}}{\bot}\,\mathbf{c}(\sigma)$$

где σ – это унификатор ℓ and ℓ'.

Conflict-Driven Clause Learning:

$$\frac{\begin{array}{cc}[\ell_1]^{i}_{1} & [\ell_n]^{i}_{n} \\ \vdots\,(\sigma^1_1,\ldots,\sigma^1_{m_1}) & \vdots\,(\sigma^n_1,\ldots,\sigma^n_{m_n}) \\ & \vdots \\ & \bot\end{array}}{(\overline{\ell_1}\sigma^1_1 \vee \ldots \vee \overline{\ell_1}\sigma^1_{m_1}) \vee \ldots \vee (\overline{\ell_n}\sigma^n_1 \vee \ldots \vee \overline{\ell_n}\sigma^n_{m_n})}\,\mathbf{cl}^i$$

где σ^k_j (для $1 \leqslant k \leqslant n$ и $1 \leqslant j \leqslant m_k$) – это композиция всех подстановок, использованных на j-ом пути [a] от ℓ_k до $\bot$.

[a] Так как доказательство – это ориентированный ациклический граф, который возможно не является деревом, то может существовать несколько путей, соединяющих ℓ_k и $\bot$ в доказательстве.

Рисунок 5 – Исчисление *Conflict Resolution*

рые были использованы для достижения конфликта. Так же как и в классическом резолюционном исчислении, вывод в *Conflict Resolution* является ориентированным ациклическим графом (не обязательно деревом). Опровержение в *Conflict Resolution* – вывод $\bot$ без использования *предположений*.

Пример 26. Рассмотрим следующую формулу:

$$\{P(z) \vee Q, P(y) \vee \neg Q, \neg P(a) \vee Q, \neg P(b) \vee \neg Q\}$$

Несмотря на то, что тут невозможно применить правило *Unit Propagation Resolution*, все равно существует опровержение этой форму-

$$\dfrac{\dfrac{\dfrac{\psi_1 : [P(x)]^1 \qquad \xi_1 : \neg P(a) \vee Q}{Q}\,\mathbf{u}(\{x\backslash a\}) \qquad \dfrac{\psi_1 \qquad \neg P(b) \vee \neg Q}{\neg Q}\,\mathbf{u}(\{x\backslash b\})}{\bot}\,\mathbf{c}(\epsilon)}{\phi_1 : \neg P(a) \vee \neg P(b)}\,\mathbf{cl}^1$$

$$\dfrac{\dfrac{\dfrac{\psi_1 : [\neg P(a)]^2 \qquad P(z) \vee Q}{Q}\,\mathbf{u}(\{z\backslash a\}) \qquad \dfrac{\psi_2 \qquad \xi : P(y) \vee \neg Q}{\neg Q}\,\mathbf{u}(\{y\backslash a\})}{\bot}\,\mathbf{c}(\epsilon)}{\phi_2 : P(a)}\,\mathbf{cl}^2$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\phi_2 \qquad \phi_1}{\neg P(b)}\,\mathbf{u}(\epsilon) \qquad \xi_2}{\neg Q}\,\mathbf{u}(\{y\backslash b\}) \qquad \dfrac{\phi_2 \qquad \xi_1}{Q}\,\mathbf{u}(\epsilon)}{\bot}\,\mathbf{c}(\sigma)$$

Рисунок 6 – Доказательство опровержения формулы из примера 26

лы, показанное на рисунке 6. Существует более короткое доказательство, но ради содержательности примера здесь показано более длинное доказательство.

1.4. Примеры актуальных задач

В данном разделе будут приведены две задачи, решаемые системами автоматического доказательства теорем: теорема из математического анализа и задача верификации аппаратного обеспечения.

1.4.1. Теорема о промежуточном значении

В качестве примера теоретической задачи возьмем теорему о промежуточном значении из математического анализа:

Теорема 27. Пусть дана непрерывная функция на отрезке $f \in C([a,b])$. Пусть также $f(a) \neq f(b)$, и без ограничения общности предположим, что $f(a) = A < B = f(b)$. Тогда для любого $C \in [A,B]$ существует $c \in [a,b]$ такое, что $f(c) = C$.

Листинг 3 показывает как данная теорема аксиоматизируется в теории первого порядка.

Листинг 3 – Теорема о промежуточном значении, записанная в *TPTP-CNF* синтаксисе

```
%----Inequality axioms
cnf(reflexivity_of_less,axiom,
    ( less_than_or_equal(X,X) )).

cnf(totality_of_less,axiom,
    ( less_than_or_equal(X,Y)
    | less_than_or_equal(Y,X) )).

cnf(transitivity_of_less,axiom,
    ( less_than_or_equal(X,Z)
    | ~ less_than_or_equal(X,Y)
    | ~ less_than_or_equal(Y,Z) )).

%----Interpolation axioms
cnf(interpolation1,axiom,
    ( ~ less_than_or_equal(X,q(Y,X))
    | less_than_or_equal(X,Y) )).

cnf(interpolation2,axiom,
    ( ~ less_than_or_equal(q(X,Y),X)
    | less_than_or_equal(Y,X) )).

%----Continuity axioms
cnf(continuity1,axiom,
    ( less_than_or_equal(f(X),n0)
    | ~ less_than_or_equal(X,h(X))
    | ~ less_than_or_equal(lower_bound,X)
    | ~ less_than_or_equal(X,upper_bound) )).

cnf(continuity2,axiom,
    ( less_than_or_equal(f(X),n0)
    | ~ less_than_or_equal(Y,X)
    | ~ less_than_or_equal(f(Y),n0)
    | less_than_or_equal(Y,h(X))
    | ~ less_than_or_equal(lower_bound,X)
    | ~ less_than_or_equal(X,upper_bound) )).

cnf(continuity3,axiom,
    ( less_than_or_equal(n0,f(X))
    | ~ less_than_or_equal(k(X),X)
    | ~ less_than_or_equal(lower_bound,X)
```

```
    | ~ less_than_or_equal(X,upper_bound) )).

cnf(continuity4,axiom,
    ( less_than_or_equal(n0,f(X))
    | ~ less_than_or_equal(X,Y)
    | ~ less_than_or_equal(n0,f(Y))
    | less_than_or_equal(k(X),Y)
    | ~ less_than_or_equal(lower_bound,X)
    | ~ less_than_or_equal(X,upper_bound) )).

%----Least upper bound axioms
cnf(crossover1,axiom,
    ( less_than_or_equal(X,l)
    | ~ less_than_or_equal(X,upper_bound)
    | ~ less_than_or_equal(f(X),n0) )).

cnf(crossover2_and_g_function1,axiom,
    ( less_than_or_equal(g(X),upper_bound)
    | less_than_or_equal(l,X) )).

cnf(crossover3_and_g_function2,axiom,
    ( less_than_or_equal(f(g(X)),n0)
    | less_than_or_equal(l,X) )).

cnf(crossover4_and_g_function3,axiom,
    ( ~ less_than_or_equal(g(X),X)
    | less_than_or_equal(l,X) )).

%----Endpoints of the interval
cnf(the_interval,hypothesis,
    ( less_than_or_equal(lower_bound,upper_bound) )).

cnf(lower_mapping,hypothesis,
    ( less_than_or_equal(f(lower_bound),n0) )).

cnf(upper_mapping,hypothesis,
    ( less_than_or_equal(n0,f(upper_bound)) )).

cnf(prove_there_is_x_which_crosses,negated_conjecture,
    ( ~ less_than_or_equal(f(l),n0)
    | ~ less_than_or_equal(n0,f(l)) )).
```

TPTP-CNF синтаксис представляет собой набор cnf высказываний, каждое из которых является дизъюнктом в итоговой формуле. Каждое cnf высказывание состоит из его имени (обычно оно как-то отображает название соответсвующей аксиомы, например «reflexivity_of_less»), типа высказывания (аксиома, предположение или отрицание теоремы) и самого логического выражения. Переменные, начинающиеся с большой буквы являются неявно квантифицированными, а с маленькой буквы – константными функциональными символами.

В этой теореме дизъюнкты разбиты комментариями по группам, соответствующим определенному типу аксиом. В конце записаны три гипотезы об упорядоченности двух границ и о том что $f(a) < n_0 < f(b)$. Последнее же высказывание – отрицание теоремы, буквально говорящее, что не существует такого l, что $f(l) = n_0$.

1.4.2. Верификация сумматора с последовательным переносом

Более практическим примером является верификация компьютерного аппаратного обеспечения. Например, сумматора с последовательным переносом. На листинге 4 приведена аксиоматизация такого сумматора.

Листинг 4 – Сумматор с последовательным переносом, записанный в *TPTP-CNF* синтаксисе

```
%----Include basic diagnosis axioms
include('Axioms/HWV001-0.ax').
%----Include model of halfadder
include('Axioms/HWV001-1.ax').
%----Include model of fulladder
include('Axioms/HWV001-2.ax').
%----Composition of 1-bit adder
cnf(nbit_adder_fulladder1,axiom,
    ( ~ type(X,nbit_adder(n1))
    | type(v(n1,X),fulladder) )).

%----Connections of 1-bit adder
cnf(nbit_adder_connection_out1_out1v1,axiom,
    ( ~ type(X,nbit_adder(n1))
    | connection(out(n1,X),out(s,v(n1,X))) )).
```

```
cnf(nbit_adder_connection_outc_outcv1,axiom,
    ( ~ type(X,nbit_adder(n1))
    | connection(out(c,X),out(c,v(n1,X))) )).

cnf(nbit_adder_connection_ina1_in1v1,axiom,
    ( ~ type(X,nbit_adder(n1))
    | connection(in(a1,X),in(n1,v(n1,X))) )).

cnf(nbit_adder_connection_inb1_in2v1,axiom,
    ( ~ type(X,nbit_adder(n1))
    | connection(in(b1,X),in(n2,v(n1,X))) )).

cnf(nbit_adder_connection_inc_incv1,axiom,
    ( ~ type(X,nbit_adder(n1))
    | connection(in(c,X),in(c,v(n1,X))) )).

%----Observations
cnf(a_isa_1bit_adder,hypothesis,
    ( type(a,nbit_adder(n1)) )).

cnf(ina1_0,hypothesis,
    ( value(in(a1,a),n0) )).

cnf(inb1_0,hypothesis,
    ( value(in(b1,a),n0) )).

cnf(inc_0,hypothesis,
    ( value(in(c,a),n0) )).

cnf(out1_0,hypothesis,
    ( value(out(n1,a),n0) )).

cnf(outc_0,hypothesis,
    ( value(out(c,a),n1) )).

%----Minimal diagnosis
cnf(diagnosis_or1v1,negated_conjecture,
    ( ~ mode(or1(v(n1,a)),abnormal) )).

cnf(diagnosis_and2h1v1,negated_conjecture,
    ( ~ mode(and2(h1(v(n1,a))),abnormal) )).
```

```
cnf(diagnosis_and2h2v1,negated_conjecture,
    ( ~ mode(and2(h2(v(n1,a))),abnormal) )).
```

Здесь используется другая *TPTP-CNF* конструкция позволяющая импортировать файлы с аксиомами. Аксиомы HWV001-0.ax, HWV001-1.ax и HWV001-2.ax были вынесены в приложение А.

Как видно по высказываниям, здесь моделируется логическая схема, в которой проводятся соединения с помощью функциональных символов in и out, а тип логического блока определяется с помощью предиката type.

Выводы по главе 1

а) Проведен обзор существующих алгоритмов для решения задачи удовлетворимости булевской формулы.
б) Поставлена задача, решаемая в текущей работе и описаны основные существующие подходы к ее решению.
в) Был проведен обзор исчисления *Conflict Resolution*, на котором основывается описываемый алгоритм.
г) Приведены примеры актуальных задач.

ГЛАВА 2. АЛГОРИТМЫ АВТОМАТИЧЕСКОГО ДОКАЗАТЕЛЬСТВА ТЕОРЕМ НА ОСНОВЕ CONFLICT RESOLUTION

2.1. Проблемы обобщения CDCL на теорию первого порядка

Обобщение *CDCL* на теорию первого порядка привносит множество проблем для любого метода доказательства основанного на распространении литералов и предположений. Далее будут рассмотрены основные сложности, с которыми пришлось столкнуться при разработке алгоритма, основанного на исчислении *Conflict Resolution*.

2.1.1. Бесконечное распространения литерала

В теории первого порядка существуют случаи, когда процедура распространения литералов может никогда не завершиться.

Рассмотрим следующий пример:

$$\{p(a), \neg p(X) \vee p(f(X)), q \vee r, \neg q \vee r, q \vee \neg r, \neg q \vee \neg r\}$$

Этот набор дизъюнктов очевидно несовместим (дизъюнкты с третьего по шестой перечисляют все возможные комбинации переменных q и r), но результатом резолюции первого и второго дизъюнктов окажется $p(f(a))$. Подобным же образом, результатом резолюции этого дизъюнкта и $\neg p(X) \vee p(f(X))$ окажется $p(f(f(a)))$. Этот процесс может продолжаться произвольно долго и генерировать бесконечный список литералов $\{p(f(a)), p(f(f(a))), \ldots, p(f(\ldots(f(a))\ldots)), \ldots\}$. Следовательно, стратегия поиска доказательства, которая будет ждать остановки процесса распространения литералов, никогда не сможет понять, что данное множество дизъюнктов несовместимо.

Похожая задача решалась при разработке различных алгоритмов доказательства в теории первого порядка и ранее. Резолюционное исчисление впервые был предложено в [5] и там же был представлен алгоритм на основе метода резолюции. Ещё тогда появились первые способы бороться с данной проблемой: такие стратегии поиска доказательств как *Set of support*, *BFS-proof-search*.

2.1.2. Отсутствие удовлетворенных литералов в удовлетворенных дизъюнктах

В пропозициональной логике удовлетворенный моделью дизъюнкт обязательно должен иметь один из своих литералов удовлетворенным в данной модели. Однако в теории первого порядка это уже не так: общие переменные создают зависимости между литералами. Более того, данный спеэффект наблюдается и во фрагменте Бернайса-Шейнфинкеля (данный термин вводится в 2.4).

Рассмотрим следующий пример:

$$\{p(X) \vee q(X), \neg p(a), p(b), q(a), \neg q(b)\}$$

Данный набор дизъюнктов является совместимым. Тем не менее не существует модели в которой $p(X)$ всегда правда (т.е. правда для всех подстановок X). Аналогично для $q(X)$. Получается, что в $p(X) \vee q(X)$ нет удовлетворенных литералов, но сам этот дизъюнкт удовлетворим.

2.1.3. Распространение литералов без удовлетворения дизъюнкта

В пропозициональной логике литерал, являющийся единственным не ложным литералом в дизъюнкте, распространяется и добавляется в модель. После этого дизъюнкт обязательно становится верным в модели и больше не должен быть рассмотрен в процедуре распространения литералов. В теории первого порядка, с другой стороны, дизъюнкт $p(X) \vee q(X)$ может распространить литерал $q(a)$ в модели, содержащей литерал $\neg p(a)$. Тем не менее $p(X) \vee q(X)$ после этого не становится верным в этой модели. Данный дизъюнкт остаётся доступным для дальнейших распространений литералов. Если, например, в модель добавится $\neg p(b)$, дизъюнкт будет вновь использован для распространения $q(b)$.

2.1.4. Квази-фальсификация без распространения

Определение 28. Дизъюнкт называется *квази-фальсифицированным* моделью M тогда и только тогда, когда все литералы этого дизъюнкта за исключением одного являются ложными в этой модели.

В теории первого порядка, в отличие от пропозициональной логики, дизъюнкт не обязательно распространит литерал когда только лишь один его литерал не является ложью. Например дизъюнкт

$p(X) \vee q(X) \vee r(X)$ квази-фальсифицирован в модели содержащей $\neg p(a)$ и $\neg q(b)$. Тем не менее никакой инстанс $r(X)$ не может быть распространен.

2.1.5. Принятие решения во время распространения литерала

Так как в *Conflict Resolution* отсутствует настоящее правило резолюции, а присутствует только лишь правило единичной резолюции, то мы не можем решить с помощью него такие случаи как:

$$\{p(a) \vee p(b), \neg p(a) \vee p(X), p(Y) \vee \neg p(b), \neg p(a) \vee \neg p(b)\}$$

Мы не можем провести ни одну резолюцию, так как здесь нет единичных дизъюнктов. Тем не менее у нас есть другое правило – правило вывод конфликтного дизъюнкта, которое как раз решает данную проблему. Однако, в отличие от классического алгоритма *CDCL*, мы не можем дождаться завершения процедуры распространения литерала, и после этого принять решение. Процедура распространения литерала может продолжаться произвольно долго даже при сбалансированной стратегии резолюций.

Рассмотрим следующий пример:

$$\{q(a) \vee r(a), q(a) \vee \neg r(a), \neg q(a) \vee \neg r(a), \neg q(a) \vee r(a), p(a), \neg p(X) \vee p(f(X))\}$$

Первые четыре дизъюнкта очевидно создают противоречие. Но система во время процедуры распространения литералов может потенциально резольвировать $p(f(a)), p(f(f(a))), \ldots$ и так далее (как было показано в 2.1.1). Для того чтобы решить данную задачу, необходимо, например, сделать предположение $q(a)$. Первые два дизъюнкта будут удовлетворены, тогда как с помощью третьего и четвертого выведутся литералы $\neg r(a)$ и $r(a)$, приводящие к конфликту. Из конфликта с помощью правила *CDCL* можно получить $\neg q(a)$ и вновь прийти к конфликту, но в этот раз используя первые два дизъюнкта. Таким образом, можно показать неудовлетворимость данной формулы.

2.1.6. Ленивые структуры данных

Одной из ключевых техник в *CDCL* является использование ленивых структур данных, таких как техника двух наблюдаемых литералов (*англ.* two watched literals) [18].

Однако есть некоторые сложности с перенесением данной техники на теорию первого порядка. Во-первых, в пропозициональном *CDCL* необходимо уметь выдавать по литералу ℓ все дизъюнкты, у которых в качестве одного из смотрителей выступает данный литерал. Но в теории первого порядка литералов ℓ', имеющих общий унификатор с данным литералом ℓ, может быть бесконечно много (например, можно представить себе ситуацию, когда для литерала $\ell = p(X)$ существуют литералы $p(a), p(f(a)), p(f(f(a))), \ldots$). Во-вторых, даже после нахождения какого-то литерала ℓ', унифицирующегося с ℓ, совершенно не факт, что после этого можно удалить ℓ из дизъюнкта.

Рассмотрим следующий пример:

$$\{\neg p(X) \vee q(X), p(a), p(b)\}$$

Тут после вывода $q(a)$ из первых двух дизъюнктов, первый дизъюнкт может быть повторно использован для вывода еще и $q(b)$.

2.2. Построение модели в теории первого порядка и поиск доказательств

Несмотря на фундаментальную разницу между пропозициональной логикой и логикой первого порядка, в текущей работе будут представлены алгоритмы первого порядка, использующие идеи алгоритма *CDCL* в логике первого порядка, при этом придерживающиеся схемы алгоритма, описанного в 1.1.3. Алгоритмы, которые будут описанны в данной главе, основаны на различных соображениях. Тем не менее, некоторые проблемы, представленные выше, они решают единым образом. В этом разделе будут описаны решения данных проблем.

Так же как и в пропозициональном случае, модел в теории первого порядка является списком литералов, но литералы здесь могут содержать универсальные (квантифицированные) переменные. Если литерал $\ell[X]$ находится в модели M, то любая его подстановка $\ell[t]$ также являет-

ся верной в M. Заметим, что проверка верности литерала ℓ в модели M гораздо дороже в логике первого порядка, нежели в пропозициональной логике: в последней нужно всего лишь проверить вхождение ℓ в M как в множество, тогда как в первой необходимо найти литерал ℓ' в M и подстановку σ такую, что $\ell = \ell'\sigma$.

Определение 29. Литерал ℓ называется *сильно верным* в модели M тогда и только тогда, когда ℓ содержится в M.

Приведем решение к проблеме 2.1.2 (отсутствие удовлетворенных литералов в удовлетворенных дизъюнктах): дизъюнкт удовлетворяется моделью M тогда и только тогда, когда все его релевантные подстановки имеют литерал верный в M. Подстановка называется релевантной если она заменяет квантифицированные переменные термами, которые встречаются в M. Таким образом, дизъюнкт $p(X) \vee q(X)$ является удовлетворенным моделью $[\neg p(a), p(b), q(a), \neg q(b)]$, так как обе релевантные подстановки $p(a) \vee q(a)$ и $p(b) \vee q(b)$ имеют литералы, которые верны в этой модели.

Основной минус такого подхода – его высокая ресурсоемкость. Для проверки одного дизъюнкта в модели M требуется генерация экспоненциального (от размера M) числа подстановок. К счастью, во многих случаях удовлетворенный дизъюнкт имеет литерал, который верен в данной модели M. Тогда дизъюнкт называется *однородно удовлетворенным*. Проверка свойства однородной удовлетворенности требует гораздо меньше ресурсов чем проверка свойства обычной удовлетворенности.

2.3. Алгоритм Propagating Depth Conflict Resolution

2.3.1. Описание алгоритма

Одним из способов бороться с бесконечным распространением литералов является *ограничение глубины доказательства*. Опишем итеративный алгоритм формально:

Пусть Δ_i – множество всех верных литералов на уровне i.

— Δ_0 состоит только из литералов, которые содержались в единичных дизъюнктах из входной формулы.

— Для всех других уровней i верно, что $\Delta_i = \Delta_{i-1} \cup \Gamma$, где Γ – результат процесса распространения литералов всех неединичных дизъюнктов входной формулы на литералах из Δ_i.

Таким образом, алгоритм не будет зацикливаться только на распространении какого-то подмножества дизъюнктов, если есть возможность провести распространение и над другими дизъюнктами. Однако, как можно заметить, здесь никак не используются предположения и алгоритм не является полным, так как формула, не содержащая ни одного единичного дизъюнкта (тем не менее неудовлетворимая), не сможет быть опровергнута.

Определение 30. Дизъюнкт называется *изначальным*, если он содержался в исходной формуле.

Как уже упоминалось в 2.1.5, необходимо делать предположения, даже если все еще можно запускать процесс распространения литералов. Будем поддерживать следующий инвариант на каждом уровне i: все изначальные дизъюнкты были использованы для получения хотя бы одной формулы из $\Delta_{i+1} \setminus \Delta_i$. Это, в частности, значит, что должна быть проведена резолюция всех дизъюнктов с уровня 0 со всеми возможными литералами с уровня 0 так, что их резольвента окажется на уровне 1. Действительно, если какой-то из дизъюнктов не внес свой вклад в новый уровень, то, возможно, процесс распространения литералов будет совершаться только над некоторым подмножеством дизъюнктов входной формулы, а для получения противоречия необходимо использование всех дизъюнктов формулы.

Чтобы поддержать данный инвариант, необходимо после запуска процесса распространения литералов на уровне i, выделить множество таких изначальных дизъюнктов Γ, которые не внесли свой вклад на уровень $i+1$. Выберем какой-то дизъюнкт $\gamma \in \Gamma$ и в нем литерал $\ell \in \gamma$. Предположив данный литерал ℓ, мы уменьшим размер множества Γ как минимум на 1. Но Γ может уменьшиться еще больше, так как после предположения ℓ потенциально могут появиться новые возможности для распространения литералов, использующие дизъюнкты из Γ. Последовательным применением данного метода можно добиться сведения размера множества Γ до 0.

2.3.2. Игнорирование бесполезных конфликтов

К сожалению, все еще существуют случаи, в которых алгоритм не может найти конфликт.

Рассмотрим следующий пример:

$$\{\neg q \vee r, \neg r \vee p, \neg r \vee q, \neg p \vee q \vee r, \neg p \vee \neg q, p \vee q\}$$

Здесь нельзя провести никаких резолюций. Поэтому происходит предположение переменной q, p или r. Предположим мы выбрали q: тогда $\{\neg r \vee q, \neg p \vee q \vee r, p \vee q\}$ будут удовлетворены, дизъюнкт $\{\neg q \vee r\}$ породит r и $\{\neg p \vee \neg q\}$ породит $\neg p$ (с помощью правила единичной резолюции). Но $\{\neg r \vee p\}$ не было использовано для распространения литералов в следующем уровне, поэтому необходимо сделать еще одно предположение (либо $\neg r$, либо p), любое из которых приведет к конфликту с r или $\neg p$. Таким образом, алгоритм не может перейти на следующий уровень, не получив при этом хотя бы один конфликт (случаи в которых мы изначально предполагаем p или r аналогичны).

У всех таких конфликтов есть одно общее свойство.

Определение 31. Конфликт называется *бесполезным*, если дизъюнкт, который из него выводится с помощью правила *CDCL*, уже содержится в системе.

После проведения процедуры распространения литералов, все предположения, которые привели *только* к бесполезным конфликтам, удаляются (вместе со всеми их выводами). Данная эвристика очень хорошо помогает на задачах, так как *PDCR* делает довольно большое число преждевременных предположений.

2.3.3. Доказательство полноты

Теорема 32. Алгоритм *PDCR* является полным относительно опровержения

Доказательство. Покажем, что *Conflict Resolution* симулирует другую полную относительно опровержения систему. Система $\mathcal{P}$ симулирует $\mathcal{Q}$ тогда и только тогда, когда существует преобразование любого $\mathcal{Q}$-вывода c из S в $\mathcal{P}$-преобразование c из S.

Листинг 5 – Резолюция в PDCR

```
function PDCR-Resolve(clause: Clause)
    result = {}
    unifyCandidates = clause.literals.map(literal => unifiableUnits[literal])
    for conclusionId ∈ [0 . . . clause.literals.size] do
        unifiers = unifyCandidates without unifyCandidates[conclusionId]
        literals = clause.literals without clause.literals[conclusionId]
        for unifier ∈ Unify(unifiers, literals) do
            clauseNode = reverseImplicationGraph[clause]
            unifierNodes = unifier.map(c => reverseImplicationGraph[c])
            newLiteral = Resolution(unifierNodes, clauseNode, literals)
            if decisions.contains(newLiteral) then
                RemoveDecision(newLiteral)
            else
                result = result ∪ {newLiteral}
            end if
        end for
    end for
    return result
end function
```

Листинг 6 – Процедура распространения литералов в PDCR

```
function PDCR-Propagation(Δ: CNF)
    while true do
        result = ∅
        for clause ∈ Δ do
            result = result ∪ PDCR-Resolve(clause)
        end for
        notUsedAncestors = initial clauses from Δ, which were not used in
result and were not satisfied by result
        while notUsedAncestors ≠ ∅ do
            Choose a clause clause in notUsedAncestors
            Choose a literal decisionLiteral in clause
            Create decision decisionLiteral
            for ancestor ∈ notUsedAncestors do
                result = result ∪ PDCR-Resolve(ancestor)
            end for
            Recalculate notUsedAncestors
        end while
    end while
end function
```

$$\dfrac{\dfrac{\dfrac{}{[\neg l_1], [\neg l_2], \ldots, [\neg l_{m-1}]}\,\mathbf{d} \qquad l_1 \vee l_2 \vee \ldots \vee l_m}{l_m}\,\mathbf{upr} \qquad \dfrac{}{[\neg l_m]}\,\mathbf{d}}{\dfrac{\bot}{l_1 \vee l_2 \vee \ldots \vee l_m}\,\mathbf{cdcl}}\,\mathbf{c}$$

Рисунок 7 – Вывод факторизации с помощью PDCR

$$\dfrac{\dfrac{\dfrac{}{[\neg l_1], \ldots, [\neg l_n]}\,\mathbf{d} \qquad l_1 \vee \ldots \vee l_n \vee l}{l}\,\mathbf{upr} \qquad \dfrac{\dfrac{}{[\neg l'_1], \ldots, [\neg l'_n]}\,\mathbf{d} \qquad l'_1 \vee \ldots \vee l'_n \vee \neg l}{\neg l}\,\mathbf{upr}}{\dfrac{\bot}{l_1 \vee \ldots \vee l_n \vee l'_1 \vee \ldots \vee l'_m}\,\mathbf{cdcl}}$$

Рисунок 8 – Вывод резолюции с помощью PDCR

Основная идея в том, чтобы проэмулировать любой вывод в резолюционном исчислении возможным ходом алгоритма PDCR.

Пусть ψ – вывод дизъюнкта c из множества S в резолюционном исчислении. Покажем PDCR-вывод ϕ дизъюнкта c из S индукцией:

— ψ – это просто вершина c. В этом случае ϕ это так же просто вершина c.

— Пусть ψ заканчивается факторизацией ρ. Тогда пусть ψ' – подвывод c' (предпосылка ρ). По предположению индукции существует вывод ϕ' формулы c' из S. Тогда можно построить ϕ из ϕ' с помощью процедуры, показанной на рисунке 7. Заметим, что все шаги показанные на данном рисунке являются валидными шагами алгоритма PDCR. Каждый шаг в данной схеме может произойти (алгоритм рандомизированный и откатывается к изначальному состоянию после неудачной попытки продвинуться).

— Пусть ψ заканчивается резолюцией. В этом случае существуют два подвывода ψ_1 и ψ_2 левой и правой предпосылок соотвественно. По предположению индукции есть ϕ_1 и ϕ_2, соответствующие ψ_1 и ψ_2. Тогда ψ может быть построено из ψ_1 и ψ_2 как показано на рисунке 8. Аналогичным образом все эти действия могут произойти: достаточно взглянуть на возможные выбираемые предположения в алгоритме.

2.4. Алгоритм Effectively Propositional Conflict Resolution

Другим способом решения проблем с бесконечным распространением литералов является *ограничение глубины распространяемых литералов*. Введем понятие глубины литерала:

Определение 33. *Глубиной* литерала называется высота дерева представляющего данный литерал.

Пример 34. Таким образом, любые константные функциональные символы или квантифицированные переменные имеют глубину 0, а глубина функции f, примененной к n аргументам, равняется максимальной из глубин ее аргументов плюс 1.

Предположим, что все формулы в нашей задаче обладают глубиной 1. Это, в частности, значит, что во всех формулах отсутствуют неконстантные функциональные символы. Теория первого порядка, ограниченная только до таких формул, называется *фрагментом Бернайса - Шейнфинкеля* (или *эффективно пропозициональным фрагментом теории первого порядка*).

Теорема 35. Множество литералов, полученных при процессе распространения литералов в эффективно пропозициональном фрагменте теории первого порядка, конечно.

Доказательство. Пусть F – исходная формула, $\mathcal{P}$ – множество предикатов, участвующих в F, а $\mathcal{V}$ – множество квантифицированных переменных и константных функциональных символов в F.

Заметим, что из $\mathcal{P}$ и $\mathcal{V}$ можно собрать конечное число литералов (а именно $\sum\limits_{p\in\mathcal{P}} |\mathcal{V}|^{|p|}$, где $|p|$ – арность предиката p). Также очевидно, что $\mathcal{P}$ и $\mathcal{V}$ не могут измениться после получения нового литерала с помощью процесса распространения литералов (этот факт напрямую следует из свойства корректности данного правила). Следовательно, множество всех литералов, которые можно получить с помощью правила *Unit-Propagation Resolution*, конечно.

По теореме 35 следует, что процесс распространения литералов в эффективно пропозициональном фрагменте теории первого порядка всегда остановится за конечное время. Но после остановки распространения литералов необходимо сделать предположение, которое сможет возобновить процесс распространения вновь.

Листинг 7 – Процедура распространения литералов в EPCR

```
function EPCR-Propagation(Δ: CNF)
    while true do
        result = ∅
        for clause ∈ Δ do
            result = result ∪ PDCR-Resolve(clause)
        end for
        Add result to Δ
    end while
end function
```

2.5. Алгоритм Term Depth Conflict Resolution

Подход, описанный в *EPCR*, можно перенести с эффктивно пропозиционального фрагмента теории первого порядка на полную теорию первого порядка. Для этого будем поддерживать *порог* максимальной глубины формул и формуле будет разрешено участвовать в распространении литералов только если ее глубина не выше, чем данный порог.

Теорема 36. Множество литералов, полученных при процессе распространения литералов из формул с глубиной не более чем k, конечно.

Доказательство. Пусть F – исходная формула, $\mathcal{P}$ – множество предикатов, участвующих в F, $\mathcal{V}$ – множество квантифицированных переменных и константных функциональных символов в F, а $\mathcal{F}$ – множество неконстантных функциональных символов. Докажем теорему по индукции. База была доказана в теореме 35.

Переход от k к $k+1$:

Пусть есть $\mathcal{L}_k$ – конечное множество литералов, полученных распространением литералов из формул F с глубиной не более чем k. Все возможные комбинации правила *Unit-Propagation Resolution* (*UPR*) с литералами из $\mathcal{L}_k$ глубины k или менее уже были сделаны на шаге k. Следовательно, в каждое новое применение данного правила нужно взять хотя бы один литерал глубины $k+1$. Заметим, что множество таких литералов не является подмножеством $\mathcal{L}_k$, так как они могли быть выведены правилом *UPR* на $k+1$-ом шаге и потом использованы для другого применения правила. Однако, их все же конечное число (а именно $\sum_{p \in \mathcal{P}} (|\mathcal{F}|^k * |\mathcal{V}|)^{|p|}$). Таким образом, различных применений правил, совер-

шенных на шаге $k+1$, будет конечное число. А значит и $\mathcal{L}_{k+1}$ будет конечным.

Из теоремы 36 следует, что процесс распространения литералов всегда остановится за конечное время. Таким образом, мы избавились от проблемы бесконечного распространения литералов и необходимости выбирать предположения во время процесса распространения литералов. Однако теперь стало не ясно, что делать после того как процесс распространение литералов остановится. Теперь у нас есть выбор между увеличением порога глубины литералов и добавлением нового предположения.

Самая простая из эвристик (и, что наиболее важно, не уменьшающая полноты алгоритма) – случайный выбор между этими двумя действиями.

Листинг 8 – Набросок алгоритма TDCR

```
function TDCR(Δ: CNF)
    Propagate(Δ)
    Seek for conflicts in new state
    If there are any new CDCL clauses – reset system with new derived clauses
    if Random(1) = 0 then
        Increase propagation threshold
    else
        Create a new decision
    end if
end function
```

ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ И ЕЕ ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

3.1. Программная реализация системы автоматического доказательства теорем Scavenger

В рамках работы было реализовано программное средство Scavenger, содержащее в себе реализации всех трех вышеописанных алгоритмов. Программная реализация разработанных алгоритмов автоматического доказательства теорем выполнялась на языке программирования *Scala*. Основные классы и связи между ними представлены на схемах 9, 10 и 11.

Исходный код программного средства Scavenger доступен в открытом доступе под лицензией GNU GPL 3.0: https://gitlab.com/aossie/Scavenger. На данный момент работа над системой активно продолжается в рамках программы Google Summer of Code 2017.

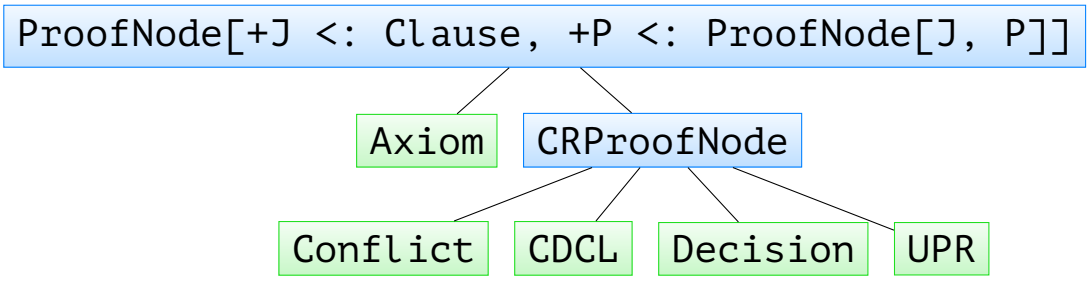

Рисунок 9 – Иерархия класса ProofNode. Синим выделены абстрактные классы, зеленым – конкретные реализации.

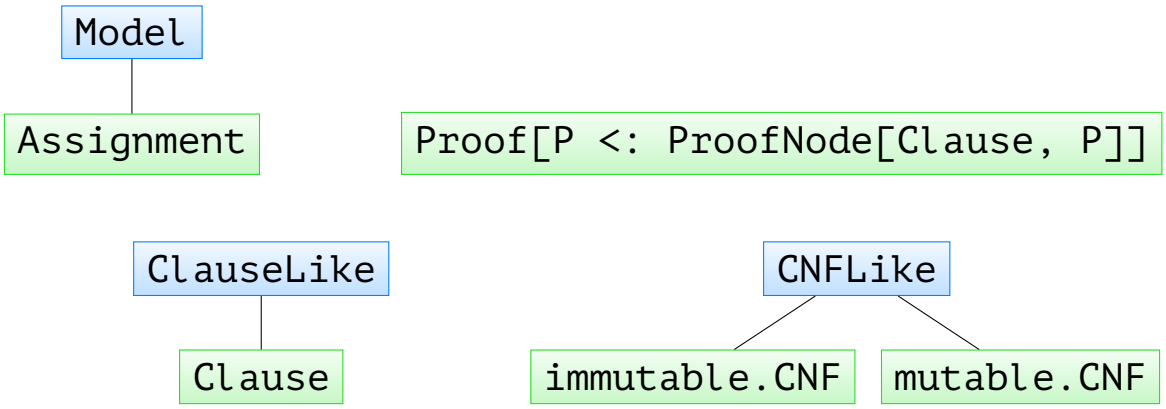

Рисунок 10 – Дополнительные структуры данных

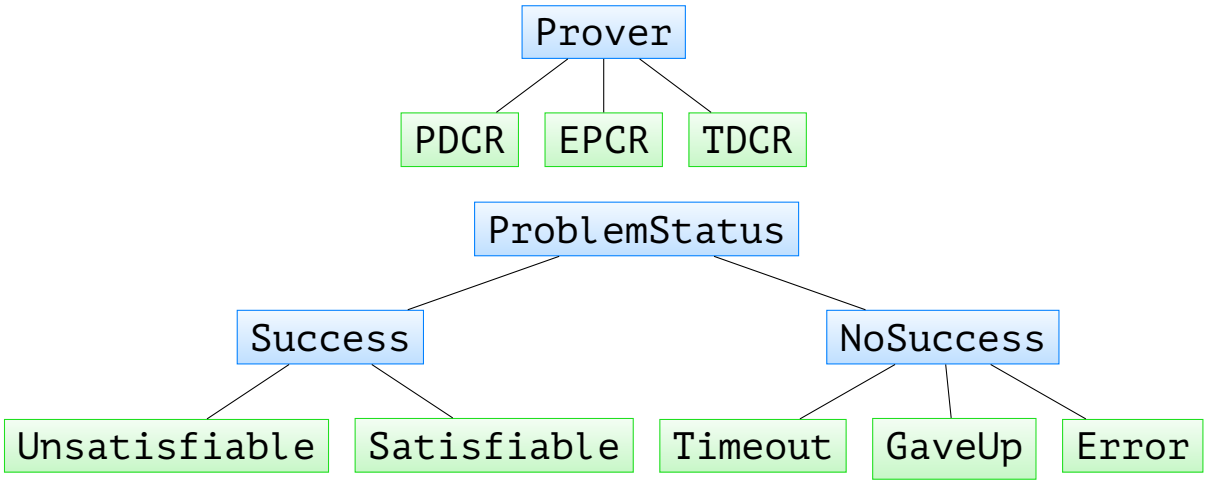

Рисунок 11 – Иерархия класса Prover и ProblemStatus

3.2. Исследование на задачах из библиотеки TPTP

Был проведен ряд экспериментальных исследований для оценки производительности предложенных алгоритмов. Был использован кластер *StarExec* [19]. В качестве источника задач использовалась библиотека *TPTP 6.4.0* [20], а конкретно *TPTP-CNF* задачи, не требующие поддержки равенства. Для сравнения была взята двадцать одна система автоматического доказательства теорем, доступных на *StarExec* и способных решать *TPTP-CNF* задачи без равенства. Для каждой рабочей пары (система и задача) было выставлено ограничение в 300 секунд времени ЦП и 600 секунд реального времени. Исследования заняли около 2 дней работы кластера из 200 машин с процессором Intel Xeon E5-2609 частотой 2.4 ГГц и 128 ГБ ОЗУ.

Таблицы показывают, сколько каждой системой было решено неудовлетворимых *TPTP-CNF* задач (общее число которых 1606) и неудовлетворимых эффективно пропозициональных задач (*ЭПЗ*) *TPTP-CNF* задач (общее число которых 572). Все варианты Scavenger-а показали себя лучше, чем *PEPR*, *GrAnDe*, *Paradox*, *ZenonModulo*, *Geo-III*, *Metis*, *Z3*, *Zipperpin* и *Otter*. TD-Scavenger решил больше задач, чем *E-Darwin* (более новая версия известной системы *Darwin*). На эффективно пропозициональном фрагменте EP-Scavenger оказался лучше 15-и других систем и решил всего на 11 задач меньше, чем система *ET*.

Система	Задач решено ЭПЗ	Все
PEPR-0.0ps	432	432
GrAnDe-1.1	447	447
Paradox-3.0	467	506
ZenonModulo-0.4.1	315	628
Geo-III-2016C	344	840
Metis-2.3	404	950
Z3-4.4.1	507	1027
Zipperpin-FOF-0.4	400	1029
Otter-3.3	362	1068
PD-Scavenger	***377***	***1082***
Bliksem-1.12	424	1107
SOS-2.0	351	1129

Система	Задач решено ЭПЗ	Все
CVC4-FOF-1.5.1	452	1145
SNARK-20120808	417	1150
Beagle-0.9.47	402	1153
EP-Scavenger	***475***	***1191***
E-Darwin-1.5	453	1213
TD-Scavenger	***467***	***1249***
Prover9-1109a	403	1293
Darwin-1.4.5	508	1357
iProver-2.5	551	1437
ET-0.2	486	1455
E-2.0	489	1464
Vampire-4.1	540	1524

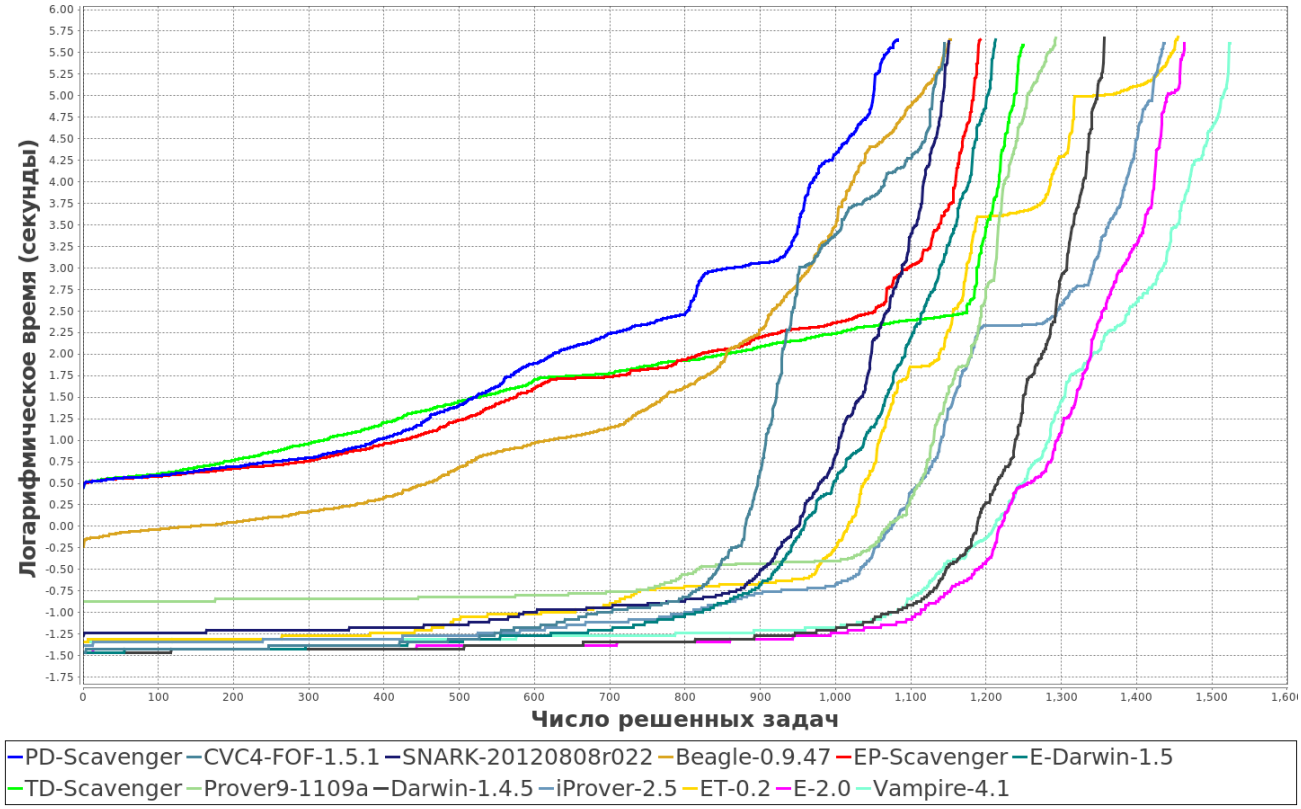

Рисунок 12 – Эффективность на неудовлетворимых *CNF* задачах из TPTP (системы отсортированы по количеству решенных задач). Системы из левой таблицы.

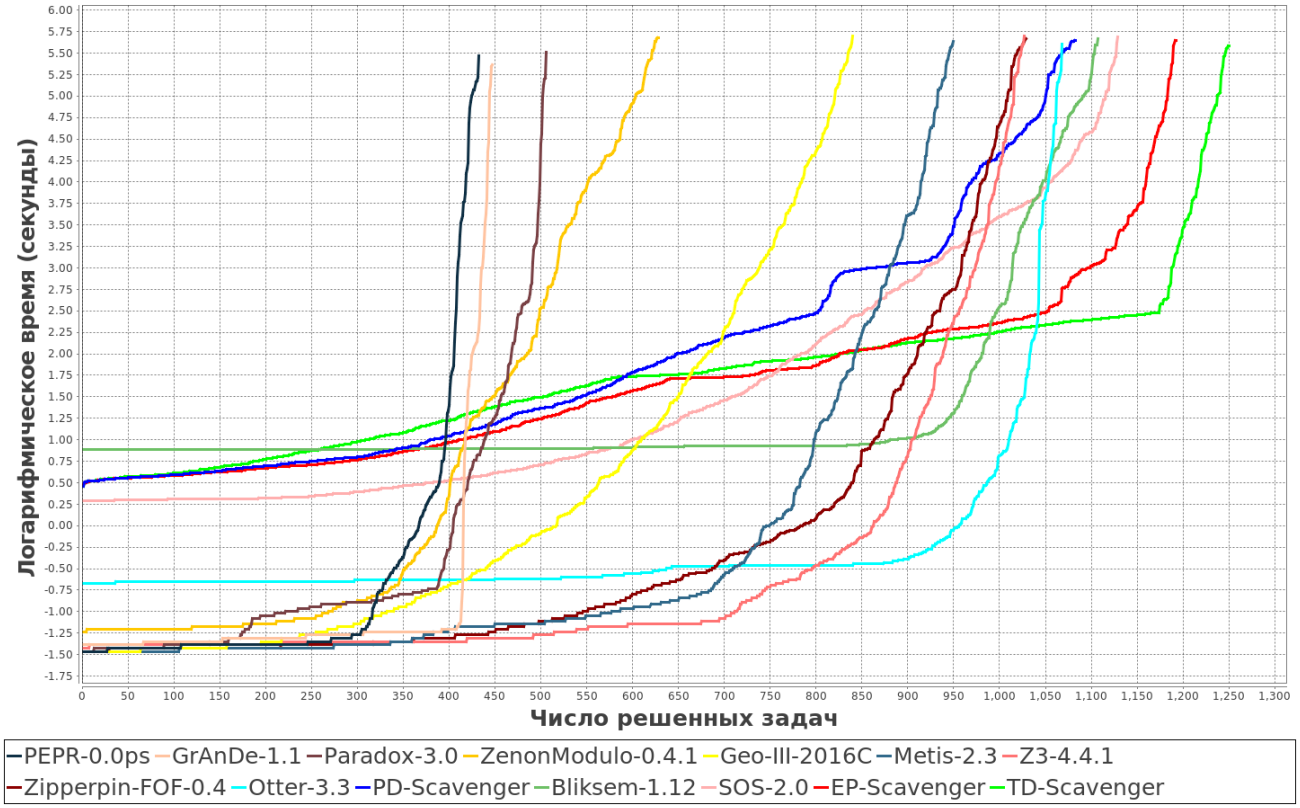

Рисунок 13 – Эффективность на неудовлетворимых *CNF* задачах из TPTP (системы отсортированы по количеству решенных задач). Системы из правой таблицы.

3.3. Исследование и сравнение трех реализаций Scavenger-а

EP-Scavenger решил на 10% больше задач, чем PD-Scavenger. Это говорит о том, что бесконечное распространение литералов – очень редкая ситуация на практике. EP-Scavenger способен решить большинство задач, несмотря на то, что он не является полным для полной теории первого порядка. TD-Scavenger решает строго больше задач, чем две другие версии Scavenger-а. Тем не менее, существует 28 задач, решить которые способен только PD-Scavenger и 26 задач, решить которые способен только EP-Scavenger.

Если говорить о сложности задач, решаемых Scavenger-ом, то текущая реализация способна решить только задачи с TPTP рейтингом не более 0.5. Scavenger хорошо справляется с задачами из теории полей и верификации аппаратного обеспечения. Список некоторых задач, решаемых Scavenger-ом, можно увидеть в приложении Б.

ЗАКЛЮЧЕНИЕ

В настоящей работе предложены несколько алгоритмов автоматического доказательства теорем в теории первого порядка. Они основаны на исчислении *Conflict Resolution* и являются обобщением идей *CDCL* на теорию первого порядка.

Работоспособность и эффективность алгоритмов была проверена на задачах из библиотеки *TPTP*. Было также проведено сравнение с существующими методами решения рассматриваемой задачи. Результаты экспериментов показывают, что несмотря на то что реализация не обладает производительностью выше, чем у таких систем как *Vampire* и *E*, у предложенных алгоритмов есть потенциал для улучшения.

Несмотря на то, что Scavenger не обладает 20-летней историей и написан на достаточно высокоуровневом языке программирования, данная система уже всерьез может состязаться с существующими системами. Предложенный в настоящем работе подход к автоматическому доказательству теорем обладает потенциалом для улучшения, чем и планируется заняться в дальнейшем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Slaney J.*, *Woltzenlogel Paleo B.* Conflict Resolution: a First-Order Resolution Calculus with Decision Literals and Conflict-Driven Clause Learning // Journal of Automated Reasoning. — 2017. — P. 1–24. — ISSN 1573-0670. — DOI: 10.1007/s10817-017-9408-6. — URL: http://dx.doi.org/10.1007/s10817-017-9408-6.

2 *Itegulov D.*, *Paleo B. W.*, *Slaney J.* Scavenger 0.1: A Theorem Prover Based on Conflict Resolution // Proceedings of the 26th International Conference on Automated Deduction (CADE). — 2017. — Forthcoming.

3 *Davis M.*, *Logemann G.*, *Loveland D.* A Machine Program for Theorem-proving // Commun. ACM. — New York, NY, USA, 1962. — July. — Vol. 5, no. 7. — P. 394–397. — ISSN 0001-0782. — DOI: 10.1145/368273.368557. — URL: http://doi.acm.org/10.1145/368273.368557.

4 *Silva J. P. M.*, *Sakallah K. A.* GRASP: A Search Algorithm for Propositional Satisfiability // IEEE Trans. Computers. — 1999. — Vol. 48, no. 5. — P. 506–521. — DOI: 10.1109/12.769433. — URL: https://doi.org/10.1109/12.769433.

5 *Robinson J. A.* A Machine-Oriented Logic Based on the Resolution Principle // J. ACM. — 1965. — Vol. 12, no. 1. — P. 23–41. — DOI: 10.1145/321250.321253. — URL: http://doi.acm.org/10.1145/321250.321253.

6 *Baumgartner P.* A First Order Davis-Putnam-Longeman-Loveland Procedure // Proceedings of the 17th International Conference on Automated Deduction (CADE). — 2000. — P. 200–219.

7 *Baumgartner P.*, *Tinelli C.* The Model Evolution Calculus // CADE. — 2003. — P. 350–364.

8 *Baumgartner P.* Model Evolution Based Theorem Proving // IEEE Inteligent Systems. — 2014. — Vol. 29, no. 1. — P. 4–10.

9 *Peter Baumgartner A. F.*, *Tinelli C.* Lemma Learning in the Model Evolution Calculus // LPAR. — 2006. — P. 572–586.

10 *Nivelle H. de*, *Meng J.* Geometric Resolution: A Proof Procedure Based on Finite Model Search // 3rd International Joint Conference on Automated Reasoning (IJCAR). — 2006. — P. 303–317.

11 *Alagi G.*, *Weidenbach C.* Non-Redundant Clause Learning // FroCoS. — 2015. — P. 69–84.

12 *Maria Paola Bonacina U. F.*, *Sofronie-Stokkermans V.* On First-Order Model-Based Reasoning // Logic, Rewriting and Concurrency. — 2015. — P. 181–204.

13 *Claessen K.* The Anatomy of Equinox – An Extensible Automated Reasoning Tool for First-Order Logic and Beyond (Talk Abstract) // Proceedings of the 23rd International Conference on Automated Deduction (CADE-23). — 2011. — P. 1–3.

14 *Korovin K.* iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description) // IJCAR. — 2008. — P. 292–298.

15 *Korovin K.* Inst-Gen - A Modular Approach to Instantiation-Based Automated Reasoning // Programming Logics. — 2013. — P. 239–270.

16 *Voronkov A.* AVATAR: The Architecture for First-Order Theorem Provers // CAV. — 2014. — P. 696–710.

17 *Brown C. E.* Satallax: An Automatic Higher-Order Prover // IJCAR. — 2012. — P. 111–117.

18 Chaff: Engineering an Efficient SAT Solver / M. W. Moskewicz [et al.] // Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001. — 2001. — P. 530–535. — DOI: 10.1145/378239.379017. — URL: http://doi.acm.org/10.1145/378239.379017.

19 *Stump A.*, *Sutcliffe G.*, *Tinelli C.* StarExec: A Cross-Community Infrastructure for Logic Solving // Automated Reasoning: 7th International Joint Conference, IJ CAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna , Austria, July 19-22, 2014. Proceedings / ed. by S. Demri, D. Kapur, C. Weidenbach. — Cham : Springer International Publishing, 2014. — P. 367–373. — ISBN 978-3-319-08587-6. — DOI:

10.1007/978-3-319-08587-6_28. — URL: http://dx.doi.org/10.1007/978-3-319-08587-6_28.

20 *Sutcliffe G.* The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0 // Journal of Automated Reasoning. — 2009. — Vol. 43, no. 4. — P. 337–362.

ПРИЛОЖЕНИЕ А. АКСИОМЫ ДЛЯ КОМПЬЮТЕРНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ HWV

Листинг А.1 – Аксиомы для соединений и базовых логических схем HWV001-0.ax

```
%----Properties of connections and values
cnf(value_propagation1,axiom,
    ( ~ connection(P1,P2)
    | ~ value(P1,V)
    | value(P2,V) )).

cnf(value_propagation2,axiom,
    ( ~ connection(P1,P2)
    | ~ value(P2,V)
    | value(P1,V) )).

cnf(unique_value,axiom,
    ( ~ value(P,V1)
    | ~ value(P,V2)
    | equal_value(V1,V2) )).

cnf(equal_value1,axiom,
    ( ~ equal_value(n0,n1) )).

cnf(equal_value2,axiom,
    ( ~ equal_value(n1,n0) )).

%----Fault model
cnf(not_ok_and_abnormal,axiom,
    ( ~ mode(K,ok)
    | ~ mode(K,abnormal) )).

cnf(ok_or_abnormal,axiom,
    ( ~ type(K,Any)
    | mode(K,ok)
    | mode(K,abnormal) )).

%----AND gate
cnf(and_0x_0,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,and)
```

```
    | ~ value(in(Any,K),n0)
    | value(out(n1,K),n0) )).

cnf(and_11_1,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,and)
    | ~ value(in(n1,K),n1)
    | ~ value(in(n2,K),n1)
    | value(out(n1,K),n1) )).

cnf(and_0_00,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,and)
    | ~ value(out(n1,K),n0)
    | value(in(n1,K),n0)
    | value(in(n2,K),n0) )).

cnf(and_1_1x,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,and)
    | ~ value(out(n1,K),n1)
    | value(in(n1,K),n1) )).

cnf(and_1_x1,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,and)
    | ~ value(out(n1,K),n1)
    | value(in(n2,K),n1) )).

%----OR gate
cnf(or_1x_1,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,or)
    | ~ value(in(Any,K),n1)
    | value(out(n1,K),n1) )).

cnf(or_00_0,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,or)
    | ~ value(in(n1,K),n0)
    | ~ value(in(n2,K),n0)
```

```
    | value(out(n1,K),n0) )).

cnf(or_1_11,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,or)
    | ~ value(out(n1,K),n1)
    | value(in(n1,K),n1)
    | value(in(n2,K),n1) )).

cnf(or_0_0x,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,or)
    | ~ value(out(n1,K),n0)
    | value(in(n1,K),n0) )).

cnf(or_0_01,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,or)
    | ~ value(out(n1,K),n0)
    | value(in(n2,K),n0) )).

%----NOT gate
cnf(not_0_1_fw,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,not)
    | ~ value(in(n1,K),n0)
    | value(out(n1,K),n1) )).

cnf(not_1_0_fw,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,not)
    | ~ value(in(n1,K),n1)
    | value(out(n1,K),n0) )).

cnf(not_0_1_bw,axiom,
    ( ~ mode(K,ok)
    | ~ type(K,not)
    | ~ value(out(n1,K),n0)
    | value(in(n1,K),n1) )).

cnf(not_1_0_bw,axiom,
```

```
    ( ~ mode(K,ok)
    | ~ type(K,not)
    | ~ value(out(n1,K),n1)
    | value(in(n1,K),n0) )).
```

Листинг А.2 – Аксиомы для полусумматора HWV001-1.ax

```
%----Composition of halfadder
cnf(halfadder_and1,axiom,
    ( ~ type(X,halfadder)
    | type(and1(X),and) )).

cnf(halfadder_and2,axiom,
    ( ~ type(X,halfadder)
    | type(and2(X),and) )).

cnf(halfadder_not1,axiom,
    ( ~ type(X,halfadder)
    | type(not1(X),not) )).

cnf(halfadder_or1,axiom,
    ( ~ type(X,halfadder)
    | type(or1(X),or) )).

%----Connections of halfadder
cnf(halfadder_connection_in1_in1or1,axiom,
    ( ~ type(X,halfadder)
    | connection(in(n1,X),in(n1,or1(X))) )).

cnf(halfadder_connection_in2_in2or1,axiom,
    ( ~ type(X,halfadder)
    | connection(in(n2,X),in(n2,or1(X))) )).

cnf(halfadder_connection_in1_in1and2,axiom,
    ( ~ type(X,halfadder)
    | connection(in(n1,X),in(n1,and2(X))) )).

cnf(halfadder_connection_in2_in2and2,axiom,
    ( ~ type(X,halfadder)
    | connection(in(n2,X),in(n2,and2(X))) )).

cnf(halfadder_connection_outs_out1and1,axiom,
```

```
    ( ~ type(X,halfadder)
    | connection(out(s,X),out(n1,and1(X))) )).

cnf(halfadder_connection_outc_out1and2,axiom,
    ( ~ type(X,halfadder)
    | connection(out(c,X),out(n1,and2(X))) )).

cnf(halfadder_connection_out1or1_in1_and1,axiom,
    ( ~ type(X,halfadder)
    | connection(out(n1,or1(X)),in(n1,and1(X))) )).

cnf(halfadder_connection_out1and2_in1not1,axiom,
    ( ~ type(X,halfadder)
    | connection(out(n1,and2(X)),in(n1,not1(X))) )).

cnf(halfadder_connection_out1not1_in2and1,axiom,
    ( ~ type(X,halfadder)
    | connection(out(n1,not1(X)),in(n2,and1(X))) )).
```

Листинг А.3 – Акиомы для сумматора HWV001-2.ax

```
%----Composition of fulladder
cnf(fulladder_halfadder1,axiom,
    ( ~ type(X,fulladder)
    | type(h1(X),halfadder) )).

cnf(fulladder_halfadder2,axiom,
    ( ~ type(X,fulladder)
    | type(h2(X),halfadder) )).

cnf(fulladder_or1,axiom,
    ( ~ type(X,fulladder)
    | type(or1(X),or) )).

%----Connections of fulladder
cnf(fulladder_connection_outsh1_in2h2,axiom,
    ( ~ type(X,fulladder)
    | connection(out(s,h1(X)),in(n2,h2(X))) )).

cnf(fulladder_connection_outch1_in2or1,axiom,
    ( ~ type(X,fulladder)
    | connection(out(c,h1(X)),in(n2,or1(X))) )).
```

```
cnf(fulladder_connection_outch2_in1or1,axiom,
    ( ~ type(X,fulladder)
    | connection(out(c,h2(X)),in(n1,or1(X))) )).

cnf(fulladder_connection_in1_in1h2,axiom,
    ( ~ type(X,fulladder)
    | connection(in(n1,X),in(n1,h2(X))) )).

cnf(fulladder_connection_in2_in1h1,axiom,
    ( ~ type(X,fulladder)
    | connection(in(n2,X),in(n1,h1(X))) )).

cnf(fulladder_connection_inc_in2h1,axiom,
    ( ~ type(X,fulladder)
    | connection(in(c,X),in(n2,h1(X))) )).

cnf(fulladder_connection_outs_outsh2,axiom,
    ( ~ type(X,fulladder)
    | connection(out(s,X),out(s,h2(X))) )).

cnf(fulladder_connection_outc_out1or1,axiom,
    ( ~ type(X,fulladder)
    | connection(out(c,X),out(n1,or1(X))) )).
```

ПРИЛОЖЕНИЕ Б. TPTP ЗАДАЧИ, РЕШАЕМЫЕ SCAVENGER-OM

Листинг Б.1 – Формат: (Имя, сложность, время потраченное на решение)

```
LCL217-1.p,0.25,21.75      LCL218-1.p,0.25,19.58
SYN716-1.p,0.25,3.9        SYN717-1.p,0.25,3.94
SYN588-1.p,0.25,2.53       SYN563-1.p,0.25,4.72
SYN557-1.p,0.25,10.67      SYN706-1.p,0.25,3.51
SYN589-1.p,0.25,2.53       SYN607-1.p,0.25,24.85
SYN705-1.p,0.25,5.24       SYN311-1.p,0.25,10.58
SYN616-1.p,0.25,29.07      SYN701-1.p,0.25,18.4
SYN608-1.p,0.25,24.77      SYN611-1.p,0.25,24.79
SYN718-1.p,0.25,4.33       SYN702-1.p,0.25,3.67
SYN646-1.p,0.25,11.1       SYN601-1.p,0.25,2.88
SYN599-1.p,0.25,50.67      SYN711-1.p,0.25,4.73
SYN704-1.p,0.25,3.78       SYN586-1.p,0.25,2.7
SYN649-1.p,0.25,3.37       SYN560-1.p,0.25,3.88
SYN661-1.p,0.25,20.03      SYN712-1.p,0.25,5.78
SYN637-1.p,0.25,3.19       SYN638-1.p,0.25,3.19
SYN606-1.p,0.25,24.77      SYN573-1.p,0.25,2.46
SYN617-1.p,0.25,99.17      SYN709-1.p,0.25,3.52
SYN703-1.p,0.25,5.05       SYN699-1.p,0.25,20.04
FLD033-3.p,0.25,41.69      NLP081-1.p,0.25,2.94
SWV337-2.p,0.25,2.98       SWV290-2.p,0.25,2.15
SWV286-2.p,0.25,2.13       COL103-2.p,0.25,1.73
MGT002-1.p,0.25,2.48       MGT017-1.p,0.25,9.37
NUM023-1.p,0.25,2.05       NUM025-1.p,0.25,2.09
NUM019-1.p,0.25,2.02       NUM020-1.p,0.25,2.04
HWV008-1.002.p,0.25,4.25   HWV006-1.p,0.25,4.72
HWV007-1.p,0.25,4.18       SET863-2.p,0.25,2.45
SET011-1.p,0.25,3.59       SET046-5.p,0.25,1.87
PUZ036-1.005.p,0.29,2.75   SYN093-1.002.p,0.33,1.94
SYN098-1.002.p,0.33,2.26   SYN097-1.002.p,0.33,2.07
SYN085-1.010.p,0.33,1.79   SYN090-1.008.p,0.33,2.19
SYN089-1.002.p,0.33,1.77   SYN094-1.005.p,0.33,2.87
SYN644-1.p,0.38,267.13     SYN687-1.p,0.38,4.64
SYN610-1.p,0.38,24.78      SYN645-1.p,0.38,282.0
SYN690-1.p,0.38,4.38       MSC001-1.p,0.38,20.73
SWV288-2.p,0.38,2.07       SWV289-2.p,0.38,2.17
SWV293-2.p,0.38,2.13       HWV008-1.001.p,0.38,3.44
SYN640-1.p,0.5,8.02        SYN639-1.p,0.5,7.89
SYN707-1.p,0.5,36.65       SYN571-1.p,0.5,4.53
SYN708-1.p,0.5,36.84       SYN647-1.p,0.5,10.92
```