ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
“ИРКУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

На правах рукописи

Ларионов Александр Александрович

Программные технологии для эффективного поиска логического вывода в исчислении позитивно–образованных формул

05.13.17 — Теоретические основы информатики

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель
кандидат технических наук
доцент Е.А. Черкашин

Иркутск — 2012

Оглавление

Введение

Актуальность исследования

Одним из подходов к интеллектуализации программных систем является разработка алгоритмов обработки информации, основанных на моделировании процесса рассуждений. Наиболее формализованные подходы базируются на автоматическом построении логического вывода (ЛВ) в некоторой системе формализованных знаний (логического описания предметной области). Программные системы для поиска ЛВ называют системами автоматического доказательства теорем (АДТ), поскольку утверждения, для которых существует ЛВ, являются теоремами (в заданном исчислении).

Формализации некоторых предметных областей, например, аксиоматизации математических теорий, верификация программного и аппаратного обеспечения и др., являются весьма громоздкими, и использование систем АДТ для построения ЛВ становится необходимым. Кроме того, системы АДТ необходимы, например, в системах планирования действий, системах поддержки принятия решений, экспертных системах, где важным фактором является время решения задачи и другие критерии. Далее, в работе, под "поиском ЛВ", как правило, будем иметь ввиду автоматизированный поиск ЛВ.

В рамках исследования формальных систем были получены следующие результаты:

1. Разрешимость логики высказываний [16]. Для любого утверждения логики высказываний существует алгоритм проверки его истинности.

2. Полуразрешимость исчислений первого порядка [71]. Если формула является теоремой, то существуют алгоритм, устанавливающий данный факт. Но если формула не является теоремой, то такого алгоритма, в общем случае, не существует.

3. Полнота исчислений первого порядка [44]. То есть соответствие между семантикой и синтаксисом, позволяющее абстрагироваться от семантики решаемой задачи.

4. Теорема Гёделя о неполноте непротиворечивых формальных систем, которые включают арифметику [45]. Принципиальная невозможность как доказательства, так и опровержения некоторых утверждений.

Хотя логика высказываний является полной и разрешимой, её язык недостаточно выразителен для решения многих проблем. С другой стороны, логики высших порядков являются весьма выразительными, но, тем не менее, часто плохо пригодными для автоматизации поиска ЛВ, поскольку в силу результатов Гёделя неэффективны и неполны. В данном случае, компромиссом являются исчисления первого порядка [41, 67, 16], достаточно выразительные для формализации многих задач, корректные, полные и полуразрешимые. Именно поэтому в основу большинства систем АДТ легли формальные системы первого порядка.

Системы АДТ появились в середине XX века, как результат развития аппарата математической логики и вычислительных систем, и успешно применяются для решения практических задач. Имеются примеры решения открытых математических проблем [55]. Более подробно об использовании систем АДТ и о современных лидерах в данной области рассказано в Главе 1.

Отметим, что важен не только сам факт установления выводимости формулы и ширина класса решаемых задач, но и затраченные вычислительные ресурсы: время решения задачи, количество шагов вывода, расход оперативной памяти. Для анализа и интерпретации полученного ЛВ важными факторами являются количество шагов вывода и способ решения, позволяющие, в том числе, получить альтернативные варианты решений задач, другие способы интерпретации результатов ЛВ. Наращивание эффективности поиска ЛВ повышает вероятность решения новых открытых проблем, даёт возможность

использовать системы АДТ в новых предметных областях. В работах С.Н. Васильева [3] развиты теоретические методы применения АДТ в управлении динамическими системами. Масштабное тестирование систем АДТ [93] показывает, что далеко не все классы задач покрыты эффективными решателями. Таким образом, проблема повышения эффективности поиска ЛВ и расширения классов решаемых задач системами АДТ является актуальной задачей. Более того, в силу полуразрешимости исчислений первого порядка, данная проблема всегда является актуальной.

Для оценки систем АДТ используются следующие формальные критерии эффективности поиска ЛВ:

1. Время решения задачи. Какое время затрачено системой АДТ для поиска ЛВ, либо установления факта невыводимости формулы.

2. Количество шагов вывода. Количество шагов, из которых состоит найденный ЛВ (количество формул в цепочке ЛВ).

3. Расход памяти. Сколько памяти компьютера затрачено системой АДТ в процессе решения задачи.

4. Ширина класса решаемых задач. Объединение класса новых задач решенных системой и класса задач, для которых предложен новый способ решения.

Очевидно, что повышение эффективности поиска ЛВ заключается в уменьшении численных характеристик критериев 1–3, и увеличении 4. Далее, в работе, говоря об эффективности, будем иметь ввиду один или несколько перечисленных критериев.

В качестве теоретического базиса разработанной, в данной диссертационной работе, системы АДТ выбрано исчисление позитивно–образованных формул (ПО–формул) [3, 4], разработанное под руководством академика Васильева С.Н., к.ф-м.н. Жерлова А.К. на основе языка типово–кванторных

формул, изначально использовавшегося для формализации свойств динамических систем и синтеза теорем в методе векторных функций Ляпунова. Опишем свойства ПО–формализма [3], в контексте АДТ и повышения эффективности поиска ЛВ, повлиявшие на выбор именного этого исчисления в качестве теоретического базиса разработанной системы:

1. Машинно–ориентированность обусловлена тем, что ПО–формулы достаточно однородны и *хорошо структурированы*, а исчисление ПО–формул включает *единственное* правило вывода и *единственную* схему аксиом.

2. Сокращение шагов ЛВ возможно благодаря крупноблочной структуре формулы и крупноблочности правила вывода. Кроме того, ПО–формулы лучше сохраняют исходную эвристическую структуру знания о задаче при переводе формализации задачи из исчисления предикатов первого порядка, чем, например, конъюнктивная нормальная форма, это позволяет использовать дополнительные структурные эвристики в процессе поиска ЛВ.

3. Сокращение расхода памяти возможно за счёт компактного представления ПО–формул и использования $\exists$–переменных в явном виде. Но эффективность данного критерия повышается в основном на уровне реализации системы АДТ.

4. Сокращение времени поиска ЛВ не заложено в само исчисление, но решается на уровне реализации системы АДТ. Тем не менее, сокращение числа шагов вывода оказывает положительное влияние на качество результата.

5. Расширение класса решаемых задач обусловлено тем, что ЛВ в исчислении ПО–формул отличается от ЛВ других исчислений, а значит возможно получение новых способов решения задач. Кроме того, выразительность языка ПО–формул позволяет использовать его более широко, в смысле предметных областей, относительно других методов АДТ.

Разработанные раннее системы АДТ, базирующиеся на исчислении ПО–формул, либо являются неэффективным, либо носят чисто демонстрационный характер [20], либо направленны на решение очень узкого класса задач. Более подробно об исчислении ПО–формул и системах, разработанных на их основе рассказано в Главе 1. Таким образом, проблема повышения эффективности поиска ЛВ актуальна и для систем АДТ, базирующихся на исчислении ПО–формул.

Объект исследования

Исчисление позитивно–образованных формул и логический вывод в данном исчислении.

Предмет исследования

Свойства языка и исчисления ПО–формул, современных структур данных и алгоритмов поиска ЛВ в исчислениях первого порядка, разработка новых структур данных и алгоритмов, обеспечивающих создание программных систем и технологий эффективного поиска ЛВ в исчислениях ПО–формул. Эффективность поиска ЛВ задается следующими критериями: время решения задачи; количество шагов ЛВ; объём использованной оперативной памяти; ширина класса успешно решаемых задач.

Цель диссертационной работы

Цель работы — разработка программной системы для эффективного поиска логического вывода в исчислении позитивно–образованных формул.

Основные задачи диссертационной работы

Для достижения описанной выше цели решаются следующие задачи:

1. Разработка эффективных структур данных представления ПО–формул в памяти компьютера.

2. Создание оригинальных методик (стратегий) эффективного поиска ЛВ в исчислении ПО–формул.

3. Адаптация существующих методик АДТ для исчисления ПО–формул с целью повышения эффективности поиска ЛВ.

4. Разработка эффективной программной системы АДТ, создание инструментальных средств для программирования систем АДТ в исчислении ПО–формул.

5. Апробация разработанных программных средств на решении тестовых и практических задач. Сравнение с другими системами АДТ.

Методика исследования

Методика исследования состоит в применении современных методов математической логики, теории программирования, информационного моделирования, методов искусственного интеллекта.

Научная новизна работы

Разработаны новые и адаптированы существующие алгоритмы, реализующие метод АДТ для исчислений ПО–формул. Использование данных алгоритмов позволило реализовать полноценную программную систему АДТ для этих исчислений. В результате проведенного исследования получены следующие новые научные результаты:

1. Изучены свойства исчисления ПО–формул, влияющих на применимость известных методик повышения эффективности поиска ЛВ.

2. Впервые предложены и реализованы подходы обработки неограниченных переменных, стратегия k, m–ограничения, структуры данных экономного представления ПО–формул с дизъюнктивным ветвлением, обработка предиката равенства без прямого представления аксиом равенства в виде ПО–формулы.

3. Впервые, для исчисления ПО–формул успешно применены широко используемые подходы: индексирование термов, параллельные схемы алгоритмов поиска ЛВ, разделение данных, кэширование результатов, системы переписывания термов.

4. Значительно расширен класс решаемых задач при помощи систем АДТ, базирующихся на исчислении ПО–формул. Построены новые варианты решения задач АДТ по сравнению с другими методами АДТ.

Научная и практическая значимость

1. Созданы система АДТ и инструментальная среда разработки систем АДТ, направленные на построение ЛВ формул исчисления ПО–формул первого порядка.

2. Выделены классы задач, для которых созданная система является более эффективной, чем современные высокопроизводительные системы АДТ. Предложены специальные стратегии для этих классов, повышающие эффективность поиска ЛВ.

3. Реализована инфраструктура тестирования разработанных алгоритмов и программного обеспечения АДТ на тестовых задачах TPTP.

Работы и исследования, проведенные в рамках диссертационной работы, выполнены в ФГБОУ ВПО “Иркутский государственный университет” и в Институте динамики систем и теории управления СО РАН.

Исследования поддержаны федеральной целевой программой, базовым финансированием СО РАН и грантами:

1. Федеральная целевая программа “Научные и научно–педагогические кадры инновационной России” на 2009-2013 годы, госконтракт № П696;

2. Базовые проекты научно–исследовательской работы ИДСТУ СО РАН. Проект IV.31.2.4. “Методы и технологии разработки программного обеспечения для анализа, обработки, и хранения разноформатных междисциплинарных данных и знаний, основанные на применении декларативных спецификаций форматов представления информации и моделей программных систем”,№ гос. регистрации: 01201001350. Программа IV.31.2. “Новые ГИС и веб–технологии, включая методы искусственного интеллекта, для поддержки междисциплинарных научных исследований сложных природных, технических и социальных систем с учетом их взаимодействия”, . Проект IV.32.1.2. Информационно–вычислительные системы нового поколения для формаций автономных необитаемых аппаратов, № гос. регистрации: 01201001346;

3. РФФИ 08-07-98005-р_сибирь_а “Программные технологии логико–математического моделирования динамики лесных ресурсов Байкальского региона”;

4. Программа “Университетский кластер”.

Разработанные программные средства используются в учебном процессе в Институте математики, экономики и информатики Иркутского государственного университета (ИМЭИ ИГУ), Национальном исследовательском

Иркутском государственном техническом университете (НИ ИрГТУ) и в научной деятельности Института динамики систем и теории управления СО РАН (ИДСТУ СО РАН).

Результаты, выносимые на защиту

1. Разработаны новые программные методики и технологии эффективного поиска ЛВ в исчислении ПО–формул: специальные структуры представления ПО–формул, стратегии поддержки неограниченных переменных, k, m–ограничение, алгоритмическая поддержка предиката равенства без явного использования аксиом равенства. Разработанные методики реализуют полезные свойства исчислений ПО–формул в программных системах АДТ.

2. Успешно адаптированы и применены существующие методики повышения эффективности поиска ЛВ для исчисления ПО–формул: индексирование термов, параллельные схемы алгоритмов, разделение данных для термов. Адаптация сохраняет полезные свойства исчислений ПО–формул.

3. Реализована новая версия программной системы для эффективного поиска ЛВ в исчислении ПО–формул первого порядка с функциональными символами и предикатом равенства. Создан программный инструментарий и технологии разработки специализированных версий программы АДТ, направленные на решение определенных классов задач АДТ.

4. Разработана инфраструктура тестирования алгоритмов и стратегий ЛВ для созданной программной системы АДТ на задачах из библиотеки TPTP. Выделены классы задач, которые система решает эффективнее, чем другие передовые системы АДТ. Значительно расширен класс успеш-

но решаемых задач по сравнению с предыдущими системами АДТ, базирующимися на исчислении ПО–формул.

Представление работы

Материалы работы докладывались на: Международной конференции “Мальцевские чтения”, г.Новосибирск, 24-28 августа 2009г.; Семинаре ИДСТУ СО РАН “Ляпуновские чтения”, ИДСТУ СО РАН, г.Иркутск, 21-23 декабря 2009г.; Всероссийской конференции молодых ученых “Математическое моделирование и информационные технологии”, г.Иркутск, 15-21 марта 2010г.; Международной конференции “Облачные вычисления. Образование. Исследования. Разработки”, г.Москва 15-16 апреля 2010г.; Международном симпозиуме по компьютерным наукам в России. Семинар “Семантика, спецификация и верификация программ: теория и приложения”, г.Казань, 14-15 июня 2010г.; 4-ой Всероссийской конференции “Винеровские чтения”, г.Иркутск, 9-14 марта 2011г. 34-ом международном симпозиуме “MIPRO”, г.Опатия, Хорватия, 23-27 мая 2011г. 4-ой Всероссийской мультиконференции по проблемам управления, с. Дивноморское, 3-8 октября 2011г. 13-ой национальной конференции по искусственному интеллекту с международным участием (КИИ-2012), г.Белгород, 16-20 октября 2012г.

Публикации по теме диссертации

По теме диссертации опубликовано 14 печатных работ, в том числе 3 статьи в журналах, входящих в перечень изданий , рекомендуемых ВАК РФ [9, 12, 14], 2 статьи в сборниках научных трудов, 9 работ в сборниках трудов конференций.

Личный вклад автора

1. Разработка оригинальных стратегий эффективного поиска ЛВ в исчислении ПО–формул [9, 14, 13, 11, 51].

2. Адаптация существующих методик эффективного поиска ЛВ в исчислении ПО–формул [9, 12, 13, 51].

3. Реализация структур данных для представления ПО–формул [9, 13, 51, 11].

4. Реализация программной системы АДТ ПО–формул [12, 14, 13, 51, 11].

5. Тестирование разработанной системы АДТ на задачах из библиотеки TPTP [12, 13, 51].

Под руководством к.т.н. Черкашина Е.А. проведена разработка структур данных для представления ПО–формул с использованием разделения оперативной памяти. Совместно с Давыдовым А.В. созданы методики поиска ЛВ с использованием неограниченных переменных.

Из печатных работ, опубликованных диссертантом в соавторстве, в текст глав 2, 3 и 4 диссертации вошли те результаты, непосредственно определяющие творческий вклад автора диссертации на этапах проектирования и разработки программного обеспечения. В перечисленных публикациях все результаты по реализации и использованию программной системы, принадлежат автору.

Соответствие паспорту специальности

Материал диссертации соответствует формуле специальности 05.13.17 – Теоретические основы информатики. Диссертация посвящена исследованию процессов создания, накопления и обработки информации, представленной в

виде фактов и знаний; созданию и исследованию новых моделей знаний, методов представления и обработки знаний; исследованию принципов создания и функционирования программных средств автоматизации представления обработки знаний.

В диссертации получены результаты по следующим пунктам “Области исследований”: 2. Исследование информационных структур, разработка и анализ моделей информационных процессов и структур; 8. Исследование и когнитивное моделирование интеллекта, включая моделирование поведения, моделирование рассуждений различных типов, образного мышления. Результаты диссертации имеют научное и народнохозяйственное значение для решения проблем повышения производительности обработки знаний в ПО–исчислении, они позволяют развить научные основы современных информационных технологий обработки формализованных знаний средствами вычислительной техники и в ускорении на этой основе научно-технического прогресса.

Структура и объем диссертации

Диссертация состоит из введения, четырех глав, заключения, списка литературы, приложений. Основной текст изложен на 122 страницах машинописного текста, полный объем диссертации 164 страницы. В работе содержится 14 рисунков. Список литературы содержит 95 наименований.

В главе 1. дан обзор исторических предпосылок к АДТ, методов и систем АДТ, области их применения. Приведено описание исчисления ПО–формул, являющегося теоретическим базисом разработанной системы АДТ. Выделены основные особенности данного исчисления и проблемы, требующие разрешения.

Глава 2. посвящена представлению методик повышения эффективности поиска ЛВ, разработанным стратегиям поиска ЛВ, базису программной си-

стемы. Представлен ряд стратегий, адаптированных из существующих систем АДТ и новые стратегии для исчисления ПО–формул, рассмотрен вопрос решения задач с предикатом равенства.

Глава 3. посвящена аспектам реализации системы. Представлена архитектура системы, описание подсистем, менеджер памяти, системные предикаты управления ЛВ, трансляторы с различных языков представления первопорядковых формул.

В главе 4. представлены методы и результаты тестирования системы. В частности, дано описание библиотеки TPTP, сводная таблица решенных задач из библиотеки TPTP. Приведены примеры, решенных задач и сравнение с передовыми системами АДТ. Даны комментарии по разработанным стратегиям и сделаны выводы о разработанной системе АДТ.

В заключении приводится список результатов, полученных в диссертации.

Приложение содержит, таблицы решенных задач из библиотеки TPTP, пример формализации задачи в языке ПО–формул, пример протокола вывода для решения одной из задач, акты о внедрении результатов данной диссертационной работы в учебные процессы ИМИЭ ИГУ, НИ ИрГТУ и в научную деятельность ИДСТУ СО РАН.

Благодарности

Автор благодарит к.т.н. Черкашина Е.А. за руководство диссертационной работой и помощь в подготовке рукописи, Давыдова А.В. за ценные указания в работе.

Глава 1. Теоретический базис исследования

1.1. Автоматическое доказательство теорем

1.1.1. Исторические предпосылки к АДТ

Пионерские идеи об автоматизации (механизации) рассуждений, скорее всего, высказали Раймунд Луллий (1235-1315) и позднее Готфрид Лейбниц (1646-1716) [15]. Луллий описывал некую механическую машину для выведения новых истин, а Лейбниц предложил создать формальный универсальный язык “lingua characteristica”, в котором можно было бы формулировать любые утверждения и создать для него исчисление “calculus ratiocinator”. Так исчисление могло быть механизировано решать вопрос об истинности утверждений, и это бы стало “освобождением человеческого разума от его собственных представлений о вещах”. В качестве примера Лейбниц рассматривал ситуацию, когда два участника спора, для проверки кто из них прав, переводят свои аргументы на “lingua characteristica” и потом говорят: “Calculemus!” — подсчитаем. Хотя эти идеи так и остались идеями не найдя никакого материального воплощения, фактически Лейбницом были сформулированы две основных составляющих для автоматизации рассуждений: специальный язык для записи утверждений, который ныне называют “формальным” и правила оперирования выражениями этого языка — “правила вывода”, в совокупности составляющими формальную систему; наличие механизма способного работать с данным языком, в качестве которого сейчас выступает компьютер.

Первая составляющая развивалась в контексте математической логики и оснований математики. Тут стоит указать на роль работ следующих исследователей: Август де Морган, Джордж Буль, Чарльз Пирс, как основоположники логики высказывания и исследователи в области алгебры; Готтлоб Фреге [41], [67], впервые описавший язык и исчисление предикатов (в

несколько неестественной для современного человека форме); Джузеппе Пеано, Бертран Рассел, Давид Гильберт, развили результаты Фреге, и поставили важные задачи [5]; Курт Гёдель, Алан Тьюринг, Алонзо Черч, получили результаты, касающиеся пределов возможностей формальных систем, в частности полнота [44] и неразрешимость теорий первого порядка, неполнота систем, выражающих арифметику; Альберт Туральф Сколем, показал что для данного множества истинных высказываний можно механически найти их доказательство; Жак Эрбран доказал, что для истинного математического предложения можно доказать что оно истинно и предложил метод доказательства. В совокупности с результатом Тьюринга и Черча это говорит о полуразрешимости теорий первого порядка. Идеи Эрбрана и по сей день лежат в основе многих методов.

Вторая составляющая развивалась в контексте вычислительных машин, основным толчком к созданию которых было в большей степени обусловлено выживанием в условиях второй мировой войны, можно отметить работы Джона фон Неймана, Норберта Винера, Алана Тьюринга, Конрада Цузе.

Развитие аппарата математической логики и вычислительных машин естественно привело к созданию первых работающих на практике систем АДТ: Программа М. Дэвиса в 1954 г. работающая на компьютере “Johniac”, доказала что сумма двух четных чисел есть четное число (первое доказательство математического утверждения, произведенное на компьютере) [15]; “Логик-теоретик”, разработанный А. Невелом, Г. Саймоном, Дж. К. Шоу [56], [57] в 1956 году для доказательства некоторых задач из Principia Mathematica [75], причем данная система была направлена на моделирование человеческих рассуждений; В 1958 г. Ван Хао создаёт систему, доказавшую 350 задач из “Principia Mathematica” [73].

1.1.2. Методы АДТ

Дадим краткое описание наиболее популярных методов АДТ. Подробный анализ методов АДТ дан в [62].

Началом сильного развития области АДТ явился метод резолюций [61] предложенный Дж. Робинсоном в 1965 году (работы велись совместно с Л. Уосом и Д. Карсоном). Причиной его успеха явилась достаточно хорошая пригодность формализма для реализации на компьютере, в частности однородность представления данных. Для резолютивных методов требуется предобработка формул (предваренные нормальные формы, сколемизация), а вывод основан на приведении исходного выражения к противоречию, т.е. для установления факта общезначимости формулы, доказывается противоречивость её отрицания. Данный метод и по сей день занимает доминирующее положение среди теоретического базиса систем АДТ, предложено множество стратегий улучшения эффективности вывода и реализованы одни из самых сильнейших систем АДТ (в том числе для логик высшего порядка) [89, 95, 79, 82, 86]. Более подробно о методе резолюций, его вариациях и стратегиях поиска ЛВ можно прочитать в [62, 19].

Другим популярным направлением в развитии методов АДТ явился табличный метод [37] предложенный в 1955г. Е.В. Бетом [29] и позднее упрощенный Р. М. Смальяном [66]. Для табличных методов не требуются специальные преобразования формул как для резолютивных методов, а доказательства строится как сведение задач к подзадачам. Развитие данного подхода уходит корнями в секвенциальное исчисление Генцена без сечения [43] предложенное в 1935г, задолго до появления метода резолюций. На базе данного подхода разработаны некоторые достаточно сильные системы АДТ [83, 85]. Как правило, табличный метод применяется для неклассических логик.

Ещё одним направлением методов АДТ являются так называемые обратные методы [40], менее известные, чем резолютивные и табличные. Впервые,

термин “обратный метод” был предложен Масловым С.Ю. [17] в 1964г, а предложенное исчисление было определено для первопорядковых логик без функциональных символов. Вывод строится в противовес табличному методу от подцелей к новым целям.

1.1.3. Область применения систем АДТ

Первоначально системы АДТ предназначались скорее для подтверждения возможности автоматизации рассуждений, а также для удовлетворения научного интереса авторов. Но почти полувековое развитие в данной области привело к тому что в 1994 году системой EQP была доказана открытая математическая проблема Роббинса [55], которая формулируется следующим образом: “Являются ли все алгебры Роббинса булевыми?”.

На сегодняшний день использование АДТ (с обоснованием его эффективности) замечено в следующих областях: верификация программ [59, 53, 84], синтез программ [64, 2], верификация оборудования [36, 77], обработка естественных языков [31], исследование протоколов [34], исследование безопасности информационных потоков [38], удаление представлений в базах данных [24], семантический веб [70, 93], составление расписаний [93], задачи управления [3], компьютерное зрение [32], математические проблемы [93, 55], решение проблем медицины [49, 93], и др.

Наибольшую популярность в настоящее время имеют следующие направления: верификация программных и аппаратных систем; синтез программного обеспечения; решение проблем математики (библиотека TPTP [93]); логическое программирование; дедуктивные базы данных [88].

1.1.4. Классификация систем АДТ

Мы классифицируем системы АДТ следующим образом:

1. Классические. Предназначены для доказательства теорем, формализо-

ванных языками первого порядка. Отличительной чертой является множество реализованных методик общего характера для повышения эффективности доказательства. Как правило, не предназначены для какой-либо специальной предметной области, и могут рассматриваться как универсальные. Наиболее известные из систем: Otter (больше не поддерживается) [89], Vampire [95], EP [79], SPASS [74], iProver [82] и др. В частности, с помощью EQP [80] была доказана открытая математическая проблема [55], а Vampire уже много лет является победителем турнира среди систем АДТ [90]. Кроме того, общей чертой данных систем, является использование лишь синтаксической информации о решаемой задаче, и минимальное взаимодействие с пользователем, т.е. развитие в направлении полной автоматизации решения задач.

2. Специализированные. Предназначены для заранее определенного класса задач и неклассических логик. Например, для различных алгебраических систем [93], геометрических задач [50, 93] и др. Характерны тем, что либо в принципе предназначены только для указанного класса, либо показывают хорошую эффективность только на задачах заданного класса, но при этом могут быть использованы и для решения других задач. Кроме того, к этому классу могут быть причислены системы АДТ для логик высшего порядка [92] или, например, для модальных логик [87].

3. Настраиваемые (полиморфные). Отчасти к таким системам можно отнести и системы из предыдущих классов, однако под настраиванием мы понимаем в большей степени настройку в соответствии с содержательной информацией о задаче. Как правило, такие системы представляют собой комбинацию существующих систем, например Isabelle [83] или система, предложенная в [30]. Coq [91] предлагает использование так называемых “тактик”, соединение нескольких типовых шагов вывода в один шаг. Интерактивные системы, хотя и не могут в полной мере быть автоматиче-

скими, всё же они интересны тем, что используют философию тесного взаимодействия человека и компьютера.

1.2. Язык и исчисление позитивно-образованных формул

В основе разрабатываемой системы лежит исчисление ПО–формул [3, 7, 72, 9].

Исчисление ПО–формул $\boldsymbol{JF}$ есть тройка $\langle \boldsymbol{LF}, Ax\boldsymbol{JF}, \boldsymbol{\omega} \rangle$, где $\boldsymbol{LF}$ — язык ПО–формул, $Ax\boldsymbol{JF}$ — единственная схема аксиом и $\boldsymbol{\omega}$ — единственное правило вывода $\boldsymbol{JF}$.

1.2.1. Язык позитивно–образованных формул

Будем обозначать множество всех конъюнктов как Con и положим что *конъюнкт* либо конечное множестве обычных атомов языка предикатов первого порядка либо $\boldsymbol{False}$, где $\boldsymbol{False}$ удовлетворяет условию $A \subset \boldsymbol{False}$ для любого $A \in Con$. Пустой конъюнкт обозначается как $\boldsymbol{True}$. Очевидно что если $A \in Con$ тогда $A \cup \boldsymbol{False} = \boldsymbol{False}$. Атомы любого конъюнкта, исключая $\boldsymbol{True}$ и $\boldsymbol{False}$, могут содержать переменные, константные и функциональные символы.

Определение 1 Пусть $\bar{x}$ есть множество переменных и A есть конъюнкт. Правильно построенные формулы языка ПОФ определяются следующим образом:

Выражение вида $\exists \bar{x} \colon A$ есть $\exists$–формула; выражение вида $\forall \bar{x} \colon A$ есть $\forall$–формула.

Пусть $G_1, \ldots, G_k$ являются $\exists$–формулами, тогда $\forall$–формула имеет следующий вид: $\forall \bar{x} \colon A\,(G_1, \ldots, G_k)$.

Пусть $G_1, \ldots, G_k$ являются $\forall$–формулами, тогда $\exists$–формула имеет следующий вид: $\exists \bar{x} \colon A\,(G_1, \ldots, G_k)$.

Выражение является правильно построенной ПО–формулой если оно построено только по правилам Определения 1.

Переменные из $\bar{x}$ связаны соответствующими кванторами и называются $\forall$-переменные и $\exists$-переменные, соответственно.

$\forall$-переменная которая не встречается в соответствующем конъюнкте называется *неограниченной* переменной.

Теперь определим семантику ПО–формул как семантику соответствующих формул языка предикатов первого порядка.

Определение 2 Пусть $A = \{A_1, \ldots, A_l\}$ есть конъюнкт и $\bar{x} = \{x_1, \ldots, x_n\}$ — множество переменных. Через $A^{\&}$ обозначим $A_1 \& \ldots \& A_l$, при этом $\boldsymbol{False}^{\&} = False, \boldsymbol{True}^{\&} = True$ (пропозициональные константы). Через F^{FOF} обозначим образ соответствующей ПО–формулы F в языке FOL.

Если $F = \exists \bar{x} \colon A$ то $F^{\text{FOF}} = \exists x_1 \ldots \exists x_n \left(A^{\&}\right)$.

Если $F = \forall \bar{x} \colon A$ то $F^{\text{FOF}} = \forall x_1 \ldots \forall x_n \left(A^{\&}\right)$.

Если $F = \exists \bar{x} \colon A\,(G_1, \ldots, G_k)$ то

$$F^{\text{FOF}} = \exists x_1 \ldots \exists x_n \left(A^{\&} \& \left(G_1^{\text{FOF}} \& \ldots \& G_k^{\text{FOF}}\right)\right).$$

Если $F = \forall \bar{x} \colon A\,(G_1, \ldots, G_k)$ то

$$F^{\text{FOF}} = \forall x_1 \ldots \forall x_n \left(A^{\&} \to \left(G_1^{\text{FOF}} \vee \ldots \vee G_k^{\text{FOF}}\right)\right).$$

Любая ПО–формула очевидно имеет структуру дерева. Таким образом, для удобства читаемости мы будем представлять их как древовидные структуры, а также пользоваться соответствующей терминологией: узел, корень, ветвь, лист и т.д.

Если ПО–формула F начинается с $\forall$: $\boldsymbol{True}$ узла, и каждый лист F является $\exists$–узлом, то F называется ПО–формулой в *канонической форме*. Очевидно, что любая ПО–формула F может быть приведена к каноническому виду с помощью следующих преобразований:

1. Если F не каноническая $\forall$–формула, тогда $\forall\colon \boldsymbol{True}\,(\exists\colon \boldsymbol{True}\,(F))$ есть ПО–формула начинающаяся с $\forall\colon \boldsymbol{True}$.

2. Если F есть $\exists$–формула, тогда $\forall\colon \boldsymbol{True}\,(F)$ есть ПО–формула начинающаяся с $\forall\colon \boldsymbol{True}$.

3. Если F имеет лист $\forall\bar{x}\colon A$, тогда новый узел $\exists\colon \boldsymbol{False}$ может быть добавлен как потомок.

Некоторые части ПО–формул имеют специальные названия: корневой (0-глубины) узел называется *корнем* ПО–формулы; любой узел глубины 1 называется *базой* ПО–формулы; максимальное поддерево начинающееся с узла глубины 1 называется *базовой подформулой*; любой узел глубины 2 называется *вопросом* к базе; максимальное поддерево начинающееся с узла глубины 2 называется *подформулой–вопросом*; максимальное поддерево начинающееся с узла глубины 3 называется *консеквентом*. В соответствии с определением семантики если узел чётной (нечетной) глубины имеет более чем одного потомка, то будем говорить, что этот узел имеет *дизъюнктивное ветвление* (соответственно *конъюнктивное ветвление*). Формулы гллубиной более 3 будем называть *глубокими*.

Пример 1 Рассмотрим формулу языка логики предикатов 1-ого порядка

$$F = \neg\big(\forall x\,\exists y P(x,y) \to \exists z P(z,z)\big).$$

Образ F^{PCF} формулы F в языке ПО–формул есть

$$F^{\text{PCF}} = \forall\colon \boldsymbol{True} - \exists\colon \boldsymbol{True} \begin{cases} \forall x\colon \boldsymbol{True} & - \quad \exists y\colon P(x,y) \\ \forall z\colon P(z,z) & - \quad \exists\colon \boldsymbol{False} \end{cases}.$$

1.2.2. Исчисление позитивно–образованных формул

Схема аксиом исчисления ПО–формул $\boldsymbol{JF}$ имеет следующую форму:

$$Ax\boldsymbol{JF} = \forall\colon \boldsymbol{True}\left(\exists\bar{x}_1\colon \boldsymbol{False}\left(\widetilde{\Phi}_1\right), \ldots, \exists\bar{x}_n\colon \boldsymbol{False}\left(\widetilde{\Phi}_n\right)\right).$$

В исчислении ПО–формул для того, чтобы доказать F мы будем пытаться опровергнуть её отрицание, поэтому аксиома исчисления ПО–формул — тождественно ложное высказывание. Таким образом процесс вывода в исчислении ПО–формул является процессом *опровержения*.

Определение 3 Будем говорить что вопрос $\forall\bar{y}\colon A$ к базе $\exists\bar{x}\colon B$ имеет *ответ* θ тогда и только тогда, когда θ есть подстановка $\bar{y} \to H^{\infty}$ и $A\theta \subseteq B$, где H^{∞} есть эрбранов универсум, основанный на $\exists$–переменных из $\bar{x}$, константных и функциональных символах которые встречаются в соответствующей базе. Ответ θ, будем иногда называть *ответной подстановкой*.

Определение 4 Если F имеет структуру $\forall\colon \boldsymbol{True}\left(\exists\bar{x}\colon B\left(\Phi\right), \Sigma\right)$, где Σ есть список других базовых подформул, и Φ есть список подформул–вопросов содержащий подформулу–вопрос $\forall\bar{y}\colon A(\exists\bar{z}_i\colon C_i\left(\Psi_i\right))_{i=\overline{1,k}}$, тогда заключение $\boldsymbol{\omega}F$ есть результат применения унарного правила вывода $\boldsymbol{\omega}$ к вопросу $\forall\bar{y}\colon A$ с ответом θ, и $\boldsymbol{\omega}F = \forall\colon \boldsymbol{True}(\exists\bar{x} \cup \bar{z}_i\colon B \cup C_i\theta\left(\Phi \cup \Psi_i\theta\right)_{i=\overline{1,k}}, \Sigma)$.

После соответствующего переименования некоторых переменных в каждой подформуле, выражение $\boldsymbol{\omega}F$ будет удовлетворять всем условиям правильно–построенных ПО–формул.

Любая конечная последовательность ПО–формул $F, \boldsymbol{\omega}F, \boldsymbol{\omega}^2F, \ldots, \boldsymbol{\omega}^nF$, где $\boldsymbol{\omega}^nF \in Ax\boldsymbol{JF}$ называется *опровержением* F в исчислении ПО–формул. Иногда будем использовать слово вывод вместо сслова опровержение.

Вопрос с консеквентом $\exists : \boldsymbol{False}$ называется *целевым вопросом*. Если вопрос имеет дизъюнктивное ветвление, то ответ на этот вопрос приводит к расщеплению соответствующей базовой подформулы на несколько новых.

Для ясности рассмотрим пример.

Пример 2 (Опровержение в JF)

$$F_1 = \forall\colon \boldsymbol{True} - \exists\colon S(e)(Q_1, Q_2, Q_3, Q_4);$$

$$\begin{aligned}
Q_1 &= \forall x\colon S(x) - \exists\colon A(a)\\
Q_2 &= \forall x, y\colon C(x), D(y) - \exists\colon \boldsymbol{False}\\
Q_3 &= \forall x, y\colon B(x), C(f(y)) - \exists\colon \boldsymbol{False}\\
Q_4 &= \forall x\colon A(x) - \begin{cases} \exists y\colon B(y), C(f(x)) \\ \exists\colon C(x) - \forall z\colon A(z), C(z) - \exists\colon D(f(z)) \end{cases}
\end{aligned}$$

На первом шаге вывода существует только один ответ $\{x \to e\}$ на вопрос Q_1. После применения $\boldsymbol{\omega}$ с этим ответом, формула приобретает следующий вид:

$$F_2 = \forall\colon \boldsymbol{True} - \exists\colon S(e), A(a)(Q_1, Q_2, Q_3, Q_4).$$

На втором шаге вывода существует только один ответ $\{x \to a\}$ на вопрос Q_4. После применения $\boldsymbol{\omega}$ с этим ответом, формула расщепляется, потому что Q_4 имеет дизъюнктивное ветвление. И теперь формулы имеет следующий вид:

$$F_3 = \forall : \boldsymbol{True} - \begin{cases} \exists y_1\colon S(e), A(a), B(y_1), C(f(a)) - \begin{cases} Q_1 \\ \dots \\ Q_4 \end{cases} \\ \exists\colon S(e)A(a), C(a) - \begin{cases} Q_1 \\ \dots \\ Q_4 \\ \forall z\colon A(z), C(z) - \exists\colon D(f(z)) \end{cases} \end{cases}.$$

На третьем шаге вывода первая база может быть опровергнута ответом $\{x \to y_1; y \to a\}$ на целевой вопрос Q_3. Опровергнутая база (подформула) для

удобства представления может быть удалена из списка базовых подформул.

На четвертом шаге вывода существует ответ $\{z \to a\}$ на пятый новый вопрос. И формула приобретает следующий вид:

$$F_4 = \forall\colon \boldsymbol{True} - \exists\colon S(e), A(a), C(a), D(f(a)) \begin{cases} Q_1 \\ \dots \\ Q_4 \\ \forall z\colon A(z), C(z) - \exists\colon D(f(z)) \end{cases}.$$

На пятом шаге вывода единственная база может быть опровергнута ответом $\{x \to a; y \to f(a)\}$ на целевой вопрос Q_4.

Опровержение закончено поскольку все базы опровергнуты.

Для данного исчисления доказаны его корректность и полнота [3, 7]. Кроме того, авторами перечисленных работ выделены основные особенности, которые предположительно должны влиять на повышение эффективности поиска ЛВ.

1.2.3. Свойства ПО–формул

1. Любая ПО–формула имеет *крупноблочную структуру* и только *позитивные кванторы* $\exists$ и $\forall$.

2. Хотя ПО–формулы содержат как квантор $\exists$ так и квантор $\forall$, структура же ПО–формул *простая*, *регулярная* и *предсказуемая* благодаря регулярному чередованию кванторов $\exists$ и $\forall$ по всем ветвям формулы.

3. Нет необходимости в предварительной обработке первоначальной формулы первого порядка с помощью процедуры сколемизации (удаление переменных, связанных кванторами существования). Процедура сколемизации приводит к увеличению сложности термов, а значит и всей фор-

мулы. Кроме того, данная особенность делает исчисление более человеко–ориентированным.

4. "Теоретические" кванторы $\forall x$ и $\exists x$ обычно не используются в формализации человеческих знаний, вместо этого чаще используются типовые кванторы $\forall x(A \to \sqcup)$ и $\exists x(A\&\sqcup)$ [33, 3, 18].

5. Исчисление ПО–формул содержит только одно правило вывода и это свойство (как и в случае метода резолюций) делает исчисление более машинно–ориентированным. Кроме того, правило вывода является крупноблочным, что сокращает количество шагов вывода, и делает вывод более понятным для человека.

6. Процедура ЛВ фокусируется в ближайшей окрестности корня ПО–формулы, благодаря особенностям 1, 2.

7. ЛВ может быть представлен в терминах *вопросно–ответной* процедуры [3], а не в технических терминах формального вывода (в терминах логических связок, атомов и др.). Базовый конъюнкт может быть интерпретирован как *база фактов*.

8. Имеется естественный *ИЛИ–параллелизм*.

9. Благодаря 1, 2, 6, 7 процедура ЛВ *хорошо совместима с конкретными эвристиками приложения*, а также с эвристиками общего управления выводом. Благодаря 5 доказательство состоит из крупноблочных шагов, и оно хорошо *наблюдаемо* и *управляемо*.

10. Благодаря 7, 9 доказательство может быть интерпретировано человеком.

11. Семантика исчисления ПО–формул может быть изменена без какой либо модификации аксиом или правила вывода $\boldsymbol{\omega}$. Такая модификация реализуется просто путём ограничений на применение правила вывода $\boldsymbol{\omega}$ и

позволяет трансформировать классическую семантику исчисления ПО–формул в немонотонную, интуиционистскую, и т.п. Примеры использования таких семантик представлены в [3].

1.2.4. Существующие системы АДТ

Из существующих систем АДТ для исчисления ПО–формул, выделим ряд версий системы КВАНТ [20, 21, 11], разработанных раннее Е.А. Черкашиным и его учениками, система М.И. Бутакова предназначенная для решения задач разбора LL-1–грамматик [2]. Система М.И. Бутакова конкретно предназначена для синтеза разборщика. Система Черкашина была разработана без учета функциональных символов, неограниченных переменных, работы с предикатом равенства, параллельных схем алгоритмов, стратегий экономии памяти и др. В ИДСТУ СО РАН реализованы еще две версии — Е.Сомова, ориентированная на управление техническими системами, она, фактически, поддерживала очень узкий подкласс ПО–формул и версия Ш.Б. Гулямова [6].

Данные системы относятся ко второму классу систем АДТ, согласно классификации изложенной выше. То есть, являются специализированными, и не предназначены для решения задач широкого класса.

1.3. Основные направления повышения эффективности поиска ЛВ

Для того, чтобы алгоритмы адекватно использовали возможности ПО–формализма необходимо выявить его совместимые полезные свойства, а также свойства, не совместимые с адаптируемыми алгоритмами. Положительные особенности, которые обеспечены алгоритмически в рамках данной диссертации, представлены выше.

Теперь опишем особенности данного формализма, которые необходимо разрешить, для эффективной реализации системы АДТ. Анализ исчисления ПО–формул показал три основные проблемы (с указанием влияния на критерии эффективности, изложенные во введении):

1. Поиск ответов на вопросы с неограниченными переменными. Данный поиск требует выбора подставляемого терма для данной переменной из эрбранова универсума, который, в общем случае, т.е. при наличии функциональных символов, является бесконечным (счетным) множеством. Какой именно терм необходимо выбрать — изначально неизвестно, причём формально любая подстановка является корректной, хотя и не обязательно приводящей в итоге к опровержению формулы. В данном случае, решение проблемы направлено на повышение эффективности согласно критериям 1,2.

2. Язык ПО–формул в [3] характеризуется как “достаточно однородный, но в то же время хорошо структурированный”, а соответствующее исчисление “хорошо усваивает эвристики”, т.е. базовая стратегия исчисления достаточно легко настраивается под конкретную задачу. Стоит заметить, что представление ПО–формул более разнообразно, чем, например, представление дизъюнктов, используемых в методе резолюций, в силу использования разнородных сущностей в структуре формулы: база, вопросы, консеквенты вопросов, два типа кванторов. То есть, требуются применение специальных методов доступа к данным неоднородным частям ПО–формулы, и разработка специальных структур данных представления ПО–формул. В данном случае, решение проблемы направлено на повышение эффективности согласно критериям 1–3.

3. Несмотря на то, что изначально представление формулы ИП в языке ПО–формул более компактно чем КНФ, применение правила вывода в процессе построения ЛВ при наличии дизъюнктивного ветвления, в общем

случае, приводит к бóльшему усложнению структуры формулы, чем при применении правила резолюции. Таким образом, через некоторое количество шагов вывода размер ПО–формулы может оказаться в разы больше чем размер соответствующей КНФ. Другим фактором, присущим для всех методов и систем АДТ является неограниченная планка сложности решаемых задач. Некоторые формализации изначально являются весьма крупными. Отсюда вытекает потребность в обеспечении экономного использования памяти, и работы системы в режиме полного заполнения доступной памяти. В данном случае, решение проблемы направлено на повышение эффективности согласно критерию 3.

1.4. Постановка задачи

Поскольку в данной работе рассматривается первопорядковый язык ПО–формул, некоторые из структур данных и алгоритмов имеют сходства с уже существующими системам АДТ первого порядка. Это касается структуры термов и представления подстановок. Кроме того, существуют стандартные методы решения конкретных технических задач, не зависящие от реализуемого формализма, например, параллельные схемы алгоритмов, экономия потребляемой памяти, индексирование данных. В связи с этим необходимо реализовать адаптацию существующих методик, используемых в современных эффективных системах АДТ. Адаптация предполагает, во–первых, учет особенностей ПО–формализма, как совместимых с данными алгоритмами, так и проблемных; во–вторых, решение задачи обеспечения совместимости адаптируемых алгоритмов, поскольку некоторые из них конфликтуют друг с другом при прямой независимой реализации.

Кроме адаптации, для повышения эффективности поиска ЛВ требуется разработать ряд оригинальных подходов присущих ПО–формализму.

Разработанные подходы должны быть направлены на повышение эффек-

тивности поиска ЛВ и не должны нарушать положительных особенностей ПО–формул. Особое внимание направлено на разрешение вышеперечисленных проблем.

Глава 2. Методики повышения эффективности поиска ЛВ

В этой главе рассматриваются задачи построения эффекивных структур данных для представления выражений языка ПО–формул, задача адаптации существующих алгоритмов, используемых в различных системах АДТ для повышения производительности процесса поиска ЛВ, а также разработка других специализированных алгоритмов. Задача этой главы — реализовать полезные свойства ПО–исчисления при помощи известных и оригинальных продуктивных методик поддержки различных этапов эффективного построения ЛВ. В качестве таких методик и алгоритмов выбраны: разделение данных (data sharing) [21, 60]; индексирование данных (term indexing) [65, 47, 69]; параллельные стратегии; для эффективной работы с предикатом равенства использованы результаты теории систем переписывания термов [26]; Предложен ряд стратегий работы с неограниченными переменными; обобщена и расширена стратегия k–опровержения [3, 20] до стратегии k, m–ограничения; разработана структура данных — дерево состояний вывода, позволяющая реализовать одновременно некоторые из перечисленных методик, один из подходов к разделению данных и процедуру возврата.

2.1. Структуры данных

Тегирование структур данных. В основе представления термов и атомов в системе лежит структура `Symbol`. Для дифференциации термов в этой структуре данных используются следующие теги: `ATOM` (атомарный символ), `FUNCTION` (функциональный символ), `CONSTANT` (константный символ), `AVARIABLE` (универсальная переменная), `EVARIABLE` (экзистенциальная переменная), `UHE` (неопределённый эрбранов элемент (НЭЭ), о нём более подробно рассказано ниже, в разделе про неограниченные переменные), `INTEGER` (цело-

численный символ), `STRING` (строковый символ). Данные значения теговых полей структур определяются перечислением:

```
enum SymbolType {CONSTANT, EVARIABLE, AVARIABLE, FUNCTION, ATOM, INTEGER, STRING, UHE};
```

Символ (`Symbol`). Символ – это идентификатор, используемый при построении терма. Структура `Symbol` содержит:

1. Строковое представление символа. Необходимо для удобного (читаемого) вывода термов, а значит и формул, на экран.

2. Тегирование символа, одним из перечисленных выше тегов.

3. Арность. Числовое значение, указывающее на количество аргументов, допустимое для данного символа. Для констант, переменных, числовых символов и строк, значение равно нулю.

Символ идентифицируется его адресом в оперативной памяти ЭВМ. Это значит, что два символа, хотя и могут иметь одинаковое строковое представление, но тем не менее будут разными символами. В основном это касается переменных, управляемых разными кванторами, но имеющих одинаковое строковое представление.

Обобщённый терм (`GTerm`). Название для структуры взято из [18]. Данная рекурсивная структура данных используется как для представления термов, так и для представления атомов, в виде деревьев.

Разные термы требуют соответствующих способов их обработки. Например, вместо универсальной переменной, возможно подставить другой терм, тем самым конкретизировав её, вывод на экран конкретизированной и неконкретизированной переменной отличается; обработка структур, представляющих функции, требует учёта наличия и количества аргументов; вместо НЭЭ не может быть подставлена универсальная переменная; и пр. Поэтому способ

работы с термом зависит от символа, лежащего в корне дерева, представляющего данный терм. Символ несёт необходимую информацию, в том числе тег. Структура символа определяет в терме необходимое количество памяти под аргументы или возможные конкретизации (подстановки).

Обобщенный терм представляется классической рекурсивной древовидной структурой, каждый узел которой содержит экземпляр структуры `Symbol` и массив ссылок на его аргументы. Универсальная переменная (`AVARIABLE`) и неопределённый эрбранов элемент (`UHE`) не содержат дочерних узлов, но используют единственный элемент одноэлементного массива ссылок-аргументов как ссылку на терм, который конкретизирует термы с тегом `AVARIABLE` и `UHE`. При помощи этой ссылки реализуется конкретизация (подстановка). Если эта ссылка не является `NULL`, то значит соответствующая переменная (или НЭЭ) конкретизирована некоторым термом, на который указывает ссылка. В противном случае переменная (или НЭЭ) является свободной для подстановки (неконкретизированной). Если переменная (или НЭЭ) конкретизирована, то терм представляющий (тегированный) переменную (или НЭЭ) в данный момент представляет конкретизирующий терм. При этом глубина конкретизации может быть сколько угодно большой, например, при такой подстановке $\theta_d = \{x \to h_1, h_1 \to h_2, h_2 \to e\}$, где x — универсальная переменная, h_1, h_2 это НЭЭ, а e — константа, после применения θ_d, переменная x фактически конкретизирована до константы e, и дальнейшая работа с термом x производится как с константой e (в том числе вывод на экран). Для корректной работы с конкретизациям реализован метод `GTerm get_value()`, который возвращает самую глубокую конкретизацию данного терма. Если заданный терм t не является переменной (или НЭЭ) или, в противном случае, если переменная (или НЭЭ) неконкретизирована, то значение `GTerm get_value()` совпадает со ссылкой на t. Иначе возвращается ссылка на самый глубокий терм, которым конкретизирована переменная (или НЭЭ).

Описанный выше подход к конкретизации термов необходим для того,

чтобы корректно и эффективно производить откат подстановок, а для этого необходимо сохранять информацию о том, какая переменная (или НЭЭ) была конкретизирована, и каким значением/термом. Для того, чтобы распознать конкретизацию достаточно проверить первый аргумент массива аргументов на совпадение с `NULL`, а для того, чтобы откатить подстановку, значению аргумента просто присваивается `NULL`.

Подстановка `Answer`. Подстановка есть список связей (`Binding`) вида $X \to t$. В структуре `Binding` для X и t заданы имена `left` и `right` соответственно. По сути `Binding` есть простая подстановка для одного терма. Как для `Answer` так и для `Binding` определены следующие методы:

- `apply()` — применить подстановку. В каждой связи терм `left` конкретизируется до терма `right`, т.е. аргумент `left` ссылается на `right`.
- `reset()` — сброс подстановки. Аргумент `left` являющийся переменной приводится в состояние `NULL`, а `left` являющийся НЭЭ не сбрасывается. Применяется непосредственно после шага вывода, для того чтобы освободить переменные для дальнейшего использования, но при этом сохранить НЭЭ конкретизированными.
- `full_reset()` — сброс всей подстановки, включая НЭЭ. Применяется на этапе неудачной унификации, чтобы вернуть все НЭЭ в прежнее состояние.

Отметим, что реализация системы задана таким образом, что изменения термов в подстановке, ведут изменения термов во всей формуле. Все изменения термов в формуле выполняются через подстановки.

Чанк — это односвязный список элементов типа `T`, в котором выделен его первый (`first`) и последний (`last`) элементы. При помощи чанков организуется основное хранение информации о ПО–формулах и состоянии ЛВ. Чанк

называется пустым, если он не содержит элементов, при этом `first` и `last` принимают значения NULL. Будем говорить что чанк C_1 связан с чанком C_2 слева, если узел last чанка C_1 ссылается на узел first чанка C_2, при этом C_2 связан с C_1 справа. Для связи чанков определена процедура связывания `void link(Chunk!(T) c)`, которая связывает слева текущий чанк и чанк `c`. Если чанки не пусты, то связывание производится согласно определению. Если один из чанков пуст, то при связывании его элементам `first` и `last` присваивается значение того элемента с которым он связывается. Отметим, что несколько чанков могут быть одновременно связаны с другим чанком с одной его стороны. Таким образом процедурой связывания можно построить дерево чанков, растущее от листьев.

На рисунке 2.1. представлен пример структуры, образованной чанками. Здесь изображено пять чанков. Друг с другом связаны через поле `first`

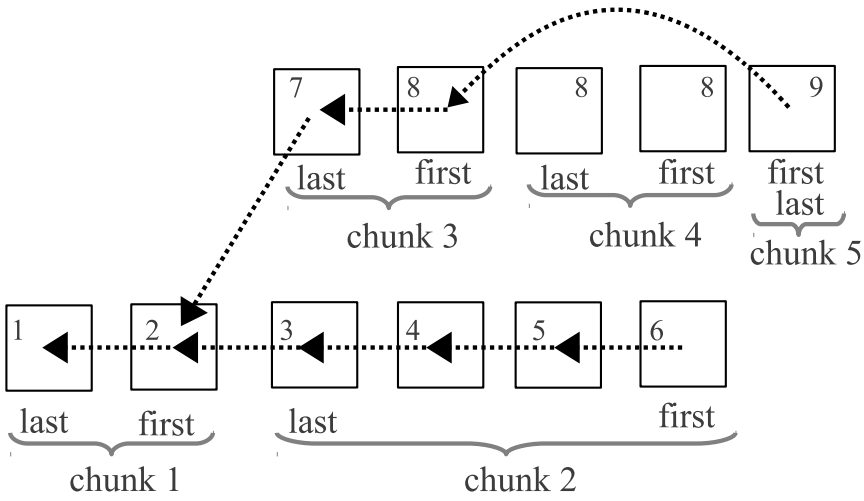

Рис. 2.1. Чанки

следующие чанки: 5 и 4, 4 и 3, 3 и 1, 2 и 1. Чанк 4 является пустым. `first` и `last` это первый и последний узлы чанков. Числа обозначают идентификатор узла. Отсюда видно что пустой чанк дублирует узел `first`. связанного слева с ним чанка.

Поясним смысл данной структуры. В отдельный чанк заносится инфор-

мация о состоянии текущего шага логического вывода, поскольку информация разнородна, то для каждого конкретного случая тип `T` конкретизируется. Связанные чанки образуют совокупность информации о шагах логического вывода. Данная структура, хотя и является довольно простой, тем не менее, позволяет реализовать и совместить многие предложенные в работе подходы.

2.1.1. Дерево состояний вывода

Одним из основополагающих средств реализации поиска ЛВ в разработанной системе АДТ является *дерево состояний вывода* (ДСВ), которое строится, в основном, при помощи структур данных, базирующихся на чанках. В структурах ДСВ, с одной стороны, хранится вся совокупность шагов вывода, по которой можно распознать какие действия были произведены на каком шаге, с другой стороны, ДСВ представляет текущую опровергаемую формулу. Основная задача ДСВ заключается в том, чтобы строго зафиксировать все события, произошедшие на каждом шаге логического вывода. Примером фиксируемого события является факт применения подстановки в базовой подформуле к некоторому вопросу. Такая фиксация событий позволяет:

1. Использовать больше информации о выполненных действиях, тем самым анализировать процесс ЛВ, а значит эффективно (в смысле большего разнообразия вариантов управления) внедрять эвристики в базовую стратегию поиска ЛВ.

2. Производить поиск ЛВ с возвратом (backtracking) в процессе построения ЛВ.

3. Реализовать стратегию разделения данных (data sharing) для случая расщепления базовых подформул после ответа на вопрос с дизъюнктивным ветвлением.

4. Производить эффективное (в смысле удобства реализации системы АДТ и производительности) управление оперативной памятью.

Идея использования ДСВ базируется на анализе свойств исчисления ПО–формул, проявляемых в процессе построения ЛВ. После каждого ответа на вопрос к первоначальной базовой подформуле добавляется пример консеквента этого вопроса: в базу добавляются соответствующие элементы узлов, непосредственно следующих за вопросом; к списку вопросов базы, в общем случае, добавляются новые вопросы; в случае дизъюнктивного ветвления база расщепляется. Таким образом, формула монотонно увеличивается, при этом сохраняя свою эвристическую структуру. ДСВ используется для обеспечения полного доступа к информации о текущем и прошлом состоянии поиска ЛВ формулы, а также для обеспечения возможности отката поиска ЛВ (backtracking) и подробного наблюдения (сбора статистики) за процессом поиска ЛВ.

Более детально, *дерево состояний вывода* есть такое дерево, которое обладает следующими свойствами: корень дерева есть одна из базовых подформул исходной ПО–формулы; все остальные узлы есть добавляемые консеквенты с применёнными к ним подстановками–ответами с необходимым разыменованием переменных. Если приводить в пример определение 4 из Главы 1 правила вывода, то корень дерева — это база $B(\Phi)$, а узлы — это $C_i(\Psi_i)\theta$. Таким образом, если происходит расщепление базы то в соответствующем узле ДСВ появляется ветвление. Теперь можно говорить, что каждая базовая подформула в формуле характеризуется соответствующим путём от листа ДСВ до её корня.

Как видно, каждый узел содержит достаточную информацию и для того, чтобы производить откат поиска, для этого достаточно просто удалять соответствующие узлы. Кроме того, такой подход реализует разделение данных (ссылок) на каждый консеквент, поскольку некоторые пути могут иметь

общие подпути. Если какая-то база опровергнута, то можно удалить все узлы от соответствующего листа до ближайшей точки ветвления, поскольку оставшаяся часть пути всё ещё используется для представления других баз. Количество листовых узлов ДСВ равно текущему количеству баз. Если дерево пусто, значит первоначальная база опровергнута. Так как изначальная формализация задачи в языке ПО–формул может содержать несколько базовых подформул, то для каждой из этих базовых подформул строится своё ДСВ.

Для практических нужд разработки специализированных версий системы АДТ узел ДСВ позволяет сохранять некоторую системную информацию:

1. Множество атомов–фактов, добавленных к базе на данном шаге вывода (который характеризуется узлом ДСВ). Данное множество представляется как чанк. Отсюда каждый базовый конъюнкт на данном шаге вывода характеризуется объединением всех чанков от данного узла до корня, при этом чанки являются связанными.

2. Список ссылок на вопросы к базе, добавленные на данном шаге вывода. Как и в случае с базовым конъюнктом, вопросы представляются связанными чанками.

3. Для каждого вопроса хранится чанк соответствующих ответов на данном шаге вывода.

4. Номер последующего шага вывода и соответствующий ответ, если узел имеет потомков.

5. Чанк использованных ответов.

6. Индексные множества (подробнее ниже в разделе про индексирование термов).

Кроме того, узлы ДСВ содержат разнородную информацию, используемую как параметры к стратегиям поиска логического вывода.

При помощи чанков получается разграничивать данные, полученные на каждом шаге, т.е. всегда возможно определить какие данные на каком шаге выводы были добавлены, и какие события произошли. Под данными имеются ввиду: атомы–факты, вопросы, ответы и т.п. С другой стороны, на каждом шаге (в каждом узле) доступны все собранные до этого данные.

ДСВ для формулы из примера 2 Главы 1 представлено на рис. 2.2.. Стрелками обозначены не связи узлов–чанков, а направление поиска ЛВ и соответственно роста ДСВ. Корнем ДСВ является исходная ПО–формула F_1. Узел 2 является консеквентом вопроса Q_1, а именно $\exists$: $A(a)$, а путь от узла 2 до корня соответствует ПО–формуле F_2. Узлы 3 и 4 соответствуют консеквентам вопроса Q_4. Путь от узла 3 до корня и путь от узла 4 до корня соответствуют базовым подформулам ПО–формулы F_3. Например, формулы определённые путями 5—1 и 3—1 разделяют данные, которые представлены узлами 1—2. Если базовая подформула, которая представлена узлами пути 3—1 опровергнута, то можно удалить путь от узла 3 до ближайшего ветвления (в сторону корня), в данном случае удаляется только узел 3, поскольку узлы 2—1 всё ещё используются для представления других базовых подформул. На рис. 2.3.

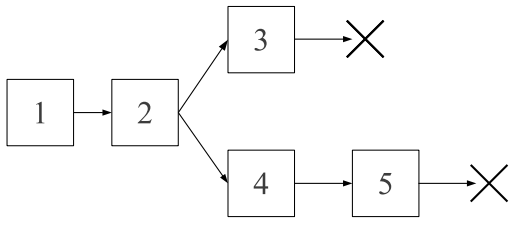

Рис. 2.2. Дерево состояний вывода для формулы из примера 2

представлено ДСВ из рис. 2.2., как связанные чанки.

Специфика связывания чанков, характеризует ДСВ как дерево направленное от листьев к корню.

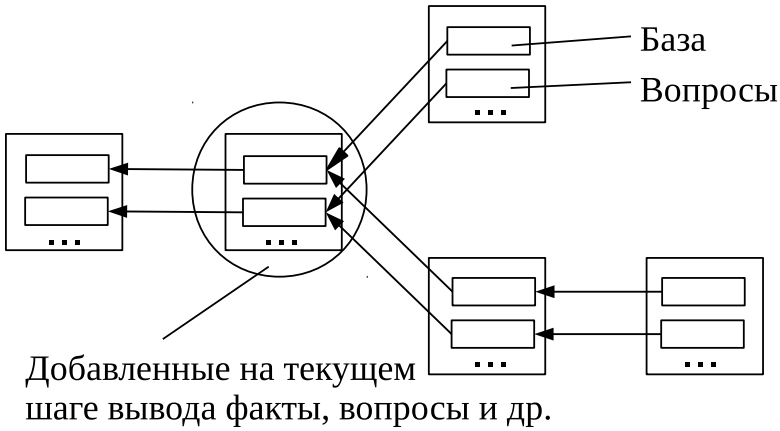

Рис. 2.3. Дерево состояний вывода рис. 2.2., представленное в виде чанков

Для ДСВ реализована процедура редукции `merge()`, преобразующая цепь узлов ДСВ в один узел. Это позволяет сократить используемую память, за счёт утраты подробной информации о шагах вывода, соответствующих цепочке узлов ДСВ.

2.2. Стратегии экономии памяти

Логический вывод практически всегда связан с получением новой дополнительной информации, ростом объема используемой оперативной памяти. Например, в методе резолюции выводятся (синтезируются) новые дизъюнкты до тех пор пока не получится пустой дизъюнкт, а в методе доказательства ПО–формул производится насыщение баз фактами до тех пор, пока все базы не станут содержать противоречие. Поскольку сложность формул может быть сколько угодно большой и даже минимальный вывод может иметь сколько угодно большую длину, имеет место проблема исчерпания имеющихся ресурсов вычислительной системы на хранение разрастающиеся структуры формулы. Опыт показывает [47], что автоматический вывод довольно быстро занимает всю имеющуюся в распоряжении оперативную память, и далее процесс вывода требует регулярное удаление излишков. За излишки можно принять любые части формулы. Например, в некоторых системах, ос-

нованных на методе резолюций, удаляются дизъюнкты, которые либо вообще не участвовали в выводе, либо не участвовали в нём определённое количество шагов, иногда удаётся определить, что дизъюнкт больше не пригодится, либо предположить, что не будет использоваться и возможно потерять полноту вывода. В случае ПО–формул, излишками являются устаревшие факты, фиктивные вопросы. Таким образом, проблема экономии оперативной памяти является основной проблемой, решаемой в диссертации, особенно с учётом увеличения сложности задач.

Для экономии памяти используются, во-первых, проектирование компактных структур данных; во-вторых, методы разделения общих участков оперативной памяти (data sharing). В случае логических языков и конкретно языка ПО–формул использование методик хранения информации с разделением общей памяти является продуктивным: экономия памяти позволяет строить как более глубокие, так и более широкие ЛВ. Исходя из некоторых общих особенностей представления языков первого порядка и представления ПО–формул, выделено и реализован ряд методик разделения данных.

Агрессивное разделение термов. Данная методика заключается в том, что разделяются общие участки оперативной памяти среди термов. Например, в термах $A(g(a, f(x)), h(c))$ и $B(k, g(a, f(x)))$, подтермы $g(a, f(x))$ являются общими и представляют собой один и тот же участок в памяти. Данный подход позволяет экономить большие объемы оперативной памяти при ограниченных ресурсах, однако требует дополнительное процессорное время на вычисление общих подтермов. Каждый новый созданный терм (например, факт, добавляемый в базу) проверяется на наличие в нём подтермов, уже используемых где-либо, т.е. производится полный поиск подтермов по базе. Если подтерм найден, то для него переносится ссылка на уже существующий. Такой метод является общеупотребимым в системах АДТ, некоторые варианты реализации представлены в [60].

Мягкое разделение термов. отличается от агрессивного подхода намного меньшим потреблением процессорных ресурсов, но и меньшей эффективностью с точки зрения объема экономии памяти, поскольку разделяет только часть общих подтермов. Исходя из определения 3 Главы 1, применение правила вывода ω корректно в случае выполнения условия $A\theta \subseteq B$, где A и B, соответственно, конъюнкты вопроса и базы. Поскольку B это уже существующее множество основных обобщенных термов, то для их хранения выделена соответствующая оперативная память. Подстановка θ же является отображением переменных вопроса A в элементы эрбранова универсума. В дальнейшем при выполнении шага вывода θ применяется (аппликируется) ко всему консеквенту вопроса, и данный консеквент добавляется к формуле. Однако правая часть подстановки уже имеется в оперативной памяти в силу того, что основана она на термах из B. Исходя из этого, достаточно использовать ссылки на структуры и уже имеющуюся память, используемые для правых частей подстановки, в тех частях консеквента, где эта подстановка применяется.

Рассмотрим следующий фрагмент базовой ПО–формулы:

$$\exists : A(f(e, g(t))) - \forall x : A(f(e, x)) - \exists : B(h(x), x)$$

В данном случае вопрос $\forall x : A(f(e, x))$ имеет ответ $\theta = \{x \to g(t)\}$. К базе фактов добавляется $B(h(x), x)\theta$, т.е. $B(h(g(t)), g(t))$. На рис. 2.4. представлен пример, демонстрирующий данную ситуацию с точки зрения мягкого разделения памяти.

Разделение базовых подформул. ПО–формулы, в которых производится ответ на вопрос с дизъюнктивным ветвлением, расщепляются на несколько новых базовых подформул. Количество новых базовых подформул совпадает с количеством непосредственных дизъюнктивных подформул в консеквенте

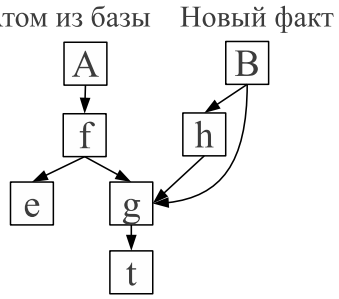

Рис. 2.4. Мягкое разделение памяти термов

вопроса. В простом варианте реализации ЛВ [20] такое расщепление требует копирования предыдущего состояния формулы несколько раз; такое копирование хотя и имеет линейную сложность [21], но всё равно естественно приводит к большим затратам памяти и процессорного времени, затрачиваемого для копирования. Разделение базовых подформул вполне реализуемо при помощи агрессивного разделения оперативной памяти термами. Однако, если формула предполагает достаточно сильное ветвление, сохраняется проблема наличия множества ссылок на разделяемые атомы баз, поскольку конъюнкт представляется как множество ссылок на атомы. Поскольку расщепление предполагает разделение общих частей баз, то имеет смысл разделять упомянутые выше ссылки. Данная стратегия реализуется за счёт средств ДСВ. Любая общая подветвь двух ветвей ДСВ является разделяемой. Рассмотрим небольшой пример. Пусть имеется следующая базовая подформула:

$$\exists : A(a) - \forall x : A(x) - \begin{cases} \exists\colon B(x) \;\; - \;\; \forall y : B(y) - \exists\colon \boldsymbol{False} \\ \exists\colon C(x) \;\; - \;\; \forall y : C(y) - \exists\colon \boldsymbol{False} \end{cases}$$

Обозначим первый и второй вопросы через Q_1 и Q_2 соответственно. Отметим, что данная формула имеет дизъюнктивное ветвление и является глубокой. Из этого следует что в соответствующем ДСВ появится ветвление, а добавленные узлы содержат не только новые факты, но и новые вопросы (в данном случае целевые). ДСВ для опровержения данной базовой подформулы пред-

ставлено на рис. 2.5..

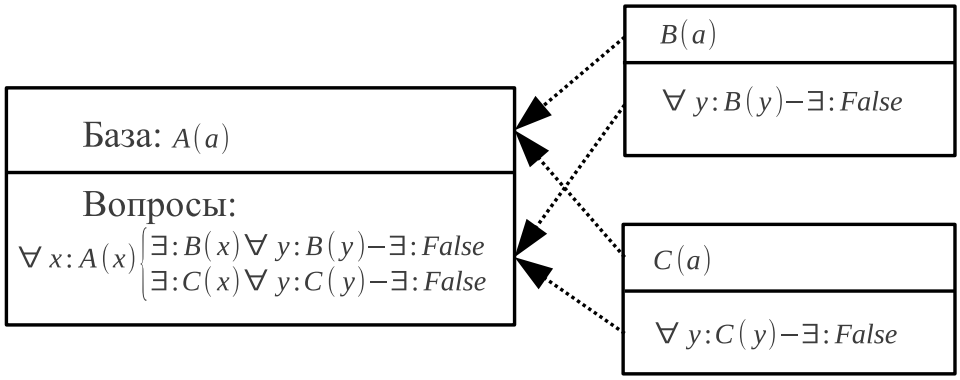

Рис. 2.5. Пример дерева состояний вывода опровержения базовой подформулы

Разделение переменных и НЭЭ. Данная методика предназначена для высокопроизводительного применения подстановки ко всей подформуле, т.к. все одноименные переменные в формуле представляют собой один участок в оперативной памяти. О неопределенных эрбрановых элементах сказано ниже. Все одноименные переменные на протяжении любого основного пути формулы [20] являются указателем на один и тот же участок в памяти. В отличие от агрессивного разделения данный подход учитывает роль переменных в процессе поиска ответных подстановок. В процессе применения подстановки переменная не заменяется на терм, а лишь указывает на этот терм, что позволяет экономить время на замену переменной термом в поддереве: достаточно одной операции присвоения над одной переменной, чтобы установить замену всех таких переменных в поддереве.

Удаление неиспользуемых фактов. Ниже, показано что процедура поиска ответных подстановок основана на том, что для каждого атома из конъюнктов вопросов производится попытка унификации с каждым атомом–фактом из базы. Поскольку, количество вопросов и длина конъюнктов конечны, то нетрудно определить для атома–факта из базы, унифицируется ли он хотя

бы потенциально с каким-либо атомом из вопросов. Если нет, то такой атом–факт можно удалить из базы, поскольку он вообще не используется в поиске ответных подстановок. Удаление ненужных фактов так же позволяет экономить память и время на переборе фактов. Особенно данная стратегия полезна при исчерпании лимита памяти, поскольку удаление фактов приводит к освобождению памяти.

Веса подформул. Под весом терма или подформулы понимается количество узлов в дереве, представляющем терм или подформулу. Анализ веса позволяет сдерживать разрастание формулы, и, соответственно, обеспечить дополнительную экономию потребляемой памяти. Для этого из возможных ответов на вопрос приоритет отдаётся тому, который приводит к формуле наименьшего веса. Под весом понимается общее количество узлов в древовидном представлении выражений. Приведем пример, пусть дана следующая простая базовая подформула с одним вопросом:

$$\exists : A(t), A(f(e,t)) - \forall x : A(x) \exists B(x)$$

В данном случае, вопрос имеет два ответа: $\theta_1 = \{x \to t\}$ и $\theta_2 = \{x \to f(e,t)\}$. В случае применения ответа θ_1 к базе добавится факт $B(t)$, а в случае применения ответа θ_2 к базе добавится факт $B(f(e,t))$. Предложенная стратегия задаёт приоритет ответу θ_1, поскольку факт $B(t)$ является выражением меньшего веса, чем факт $B(f(e,t))$. Вес факта $B(t) = 2$, а вес факта $B(f(e,t)) = 4$.

Повторы атомов и ответов. Повторное использование одинаковых ответов, и внесение в базу одинаковых фактов запрещено.

2.3. Индексирование данных.

Формализации некоторых задач могут быть довольно обширными, например задача `ALG214+4` из библиотеки TPTP в языке ПО–формул занимает порядка 10 Мб. Кроме того, даже если изначальная формула не столь велика, после некоторого количества шагов вывода, она может разрастись до достаточно большого размера. Стратегии разделения данных позволяют лишь отсрочить момент переполнения памяти, а так же вместить в имеющуюся память формулу как можно большего размера. Кроме того, запрет использования повторов требует эффективного поиска заданных выражений по формуле.

С другой стороны существуют ситуации, когда необходимо анализировать ПО–формулу на наличие определённых свойств. Такой анализ проводится как в рамках поиска ЛВ, так и отдельно. Например, если для задачи заданна некоторая стратегия вывода, то для осуществления очередного шага вывода выбирается ответ, а значит база и вопрос, соответствующий этой стратегии, т.е. анализируется вся формула, и выбираются удовлетворяющие части формулы стратегии. Кроме того, анализ формул часто требуется после вывода формулы, для сбора статистики и другой информации, позволяющей в дальнейшем разрабатывать новые стратегии, либо извлекать содержательную информацию из ЛВ.

Отсюда необходимо разработать методы, позволяющие в формуле производить эффективный поиск необходимых её элементов (термов, вопросов, ответов). Для решения данной проблемы обратимся к существующему опыту.

Индексирование термов. Основной структурой, используемой для представления формул, является обобщенный терм. Он используется для представления элементов конъюнктов, кванторных переменных, обеих частей подстановок. Поэтому актуальной задачей является эффективный поиск обоб-

щенных термов в формуле по заданным критериям.

В информационных технологиях поиск данных часто разрешается, например, с помощью методов индексирования данных, применяемых широко в реляционных базах данных БД [42]. В нашем случае основной объект индексирования — это обобщенный терм, который является древовидной структурой, индексирование которой методами, используемыми в реляционных БД, неэффективно [47]. Поэтому используются нижеописанные подходы.

Индексирование термов к настоящему времени хорошо исследовано, как в рамках определенных систем АДТ, так и абстрактно. В частности по данной теме существует ряд интересных работ, в том числе [54, **?**, 47, 65]. Представленные в этих работах методы позволяют эффективно находить в базе термов такие термы, которые удовлетворяют определенным критериям: являются равными данному (query term), являются его примерами, обобщениями (generalization) и унификациями. Для разработанной системы АДТ требуется поиск заданого терма, его основных примеров, его обобщений, и унификаций. С учетом перечисленных требований а также того факта, что как правило, в базе находятся основные термы, в качестве основы методики индексирования выбрано индексирование путями [69, 54]. Кратко опишем её суть, как она описана в [54].

Для каждого символа входящего в терм составляется список так называемых *путей*. Путь — это последовательность чередующихся символов и чисел, где число определяет позицию символа среди дочерних узлов [54]. Например, атом $A(e, f(f(x, k), y), m)$ представляется в виде путей A, $A1e$, $A2f$, $A2f1f$, $A2f1f1x$, $A2f1f2k$, $A2f2y$, $A3m$. То есть каждому символу соответствует путь от корня до этого символа в древовидном представлении обобщенного терма. Подробнее на рис. 2.6. Каждый из этих путей содержит указатель на соответствующий терм, т.е. тот терм, для которого строился путь. Сами пути хранятся в отсортированном виде.

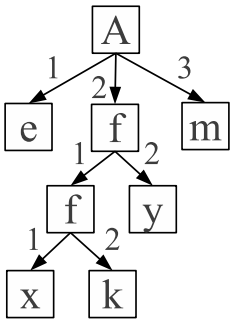

Рис. 2.6. Пример построения путей

Пример. Пусть дан атом $A(a, f(x, b))$ и множество атомов

$$\{A(a, f(c, b)), A(a, f(b, e)), A(a, f(k, b)), A(b, f(e, b))\}.$$

Любой атом, являющийся основным примером для заданного, содержит в своём списке путей следующие пути: $A1a$, $A2f2b$. Таким образом, для поиска основных примеров заданного атома необходимо найти пересечение множеств атомов, на которые указывают пути. В частности путь $A1a$ указывает на множество

$$\{A(a, f(c, b)), A(a, f(b, e)), A(a, f(k, b))\},$$

а путь $A2f2b$ на

$$\{A(a, f(c, b)), A(a, f(k, b)), A(b, f(e, b))\}.$$

Пересечение этих множеств есть множество $\{A(a, f(c, b)), A(a, f(k, b))\}$, это и есть множество примеров для атома $A(a, f(x, b))$.

Методы индексирования термов в настоящее время широко используются во многих известных системах АДТ.

Отметим следующую особенность внедрения данного подхода к разработанной системе АДТ. Поскольку в системе широко используются методы разделения данных, необходимо совместить индексные множества с подходом к разделению данных. Проблема заключается в следующем. Каждый

путь ссылается на множество соответствующих термов, однако поскольку если формула имеет дизъюнктивное ветвление, то в ходе вывода производится её расщепление и ДСВ ветвится соответствующим образом. Для каждой базовой подформулы необходим свой индекс, поскольку базовые подформулы разделяются с использованием возможностей ДСВ, индексное множество также разделяется этой структурой. И вместо списка, традиционно используемого для представления множества термов соответствующих данному пути, строится дерево чанков, в котором множества индексируемых термов соответствуют путям от листа дерева до корня, и каждый такой путь есть индексное множество для данной базовой подформулы. Таким образом, чанковым деревом для хранения индексных множеств может служить ДСВ. Поскольку совмещение метода индексирования путями с методами разделения данных базируется на совмещение индексных множеств с ДСВ, в системе, кроме того, реализуется подход к быстрому откату индекса. Отличие от традиционно используемого метода индексирования путями заключается в том что каждый путь ссылается не надо одно множество соответствующих термов, а на ряд таких множеств, каждое из которых соответствует базовой подформуле, но при этом множства разделяются возможностями ДСВ и тем самым не создают лишней нагрузки на систему (копирование индексных множеств, простые возвраты). Поясним данный подход на следующем примере. Пусть дан фрагмент базовой подформулы

$$\exists : A(e,k), B(a,b) - \forall x : B(x,y) - \begin{cases} \exists \colon A(x,k) \\ \exists \colon A(y,k) \end{cases}$$

Ответ на вопрос приведет к расщеплению базовой подформулы на две, с конъюнктами $\{A(e,k), B(a,b), A(a,k)\}$ и $\{A(e,k), B(a,b), A(b,k)\}$. ДСВ с указанием разделяемых базовых конъюнктов показано на рис. 2.7.

На рис. 2.8. представлен индекс данного множества термов с учетом раз-

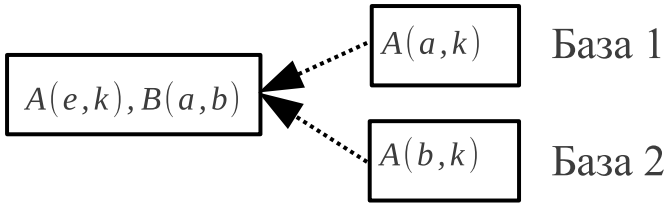

Рис. 2.7. Дерево состояний вывода с указанием разделяемых базовых конъюнктов

деления данных.

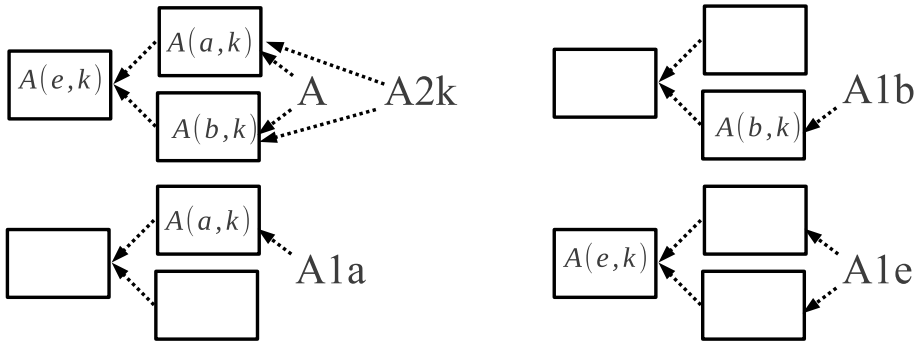

Рис. 2.8. Индексирование путями с учетом дерева состояний вывода

Индексирование других частей формулы. Кроме базы термов, в индексировании нуждаются множества вопросов и множества ответных подстановок. Поскольку количество вопросов не так велико как количество термов, то достаточно использовать возможности обычного словаря. Теперь опишем другие проблемы, решаемые при помощи методов индексирования данных. Предположим, что пользователем задана некоторая стратегия для решения некоторого класса задач. Стратегия оперирует вопросами с определенными свойствами (т.е. с вопросом сопоставляется содержательная информация), которые присущи всему классу задач (т.е. все задачи объединяет наличие определенных вопросов). Для повышения производительности поиска ЛВ необходимо упростить в дальнейшем доступ к таким вопросам, чтобы каждый раз повторно не проверять все вопросы на наличие определенного свойства.

Для этого достаточно сделать словарь (map) в котором каждому описанию вопроса соответствует указатель на данный вопрос.

Индексирование ПО–формул. Доказательство крупных формул может производиться в несколько этапов, с сохранением промежуточных результатов на жесткие носители информации. Сам логический вывод формально является цепочкой формул. Для анализа описанных ситуаций необходимо проверять формулы на наличие некоторых свойств. Поскольку, ПО–формула как и обобщенный терм имеет древовидную структуру, то для её индексирования применимы описанные выше методы индексирования путями. Однако в данном случае индексирование путями требует доработки, поскольку, в случае термов путь однозначно задавался символами и номерами дочерних узлов, то в случае ПО–формул номера дочерних узлов сохраняются, а вот символу может соответствовать любая формально описанная характеристика узла ПО–формулы (типового квантора). В системе используются следующие характеристики: тип квантора, размер конъюнкта, лексикографическое представление одного из термов конъюнкта

2.4. Неограниченные переменные

Проблема неограниченных переменных описана выше. Кратко опишем её суть. Согласно определению 3 Главы 1 правило вывода ω применимо в том случае если вопрос $\forall\bar{y}\colon A$ к базе $\exists\bar{x}\colon B$ имеет *ответ* θ такой что θ есть подстановка $\bar{y} \to H^\infty$ и $A\theta \subseteq B$. В случае если все переменные из $\bar{y}$ содержатся в конъюнкте A, то поиск соответствующих элементов из H^∞ производится с помощью алгоритма поглощения и зависит только от сооветствующей базы фактов, поскольку должно выполнятся условия поглощения множеств. Если же среди $\bar{y}$ имеются переменные, которые не входят в A, то такие переменные называются *неограниченными* и не используются в алгоритме поглощения.

Однако для этих переменных необходимо найти подстановку, причём любой элемент эрбранова универсума является формально корректной подстановкой. Проблема в том, что в общем случае (при наличии функциональных символов) эрбранов универсум является бесконечным множеством, и какой именно элемент из него необходимо выбрать неизвестно.

Данная проблема не является тривиальной. Предложено несколько подходов её решения.

2.4.1. Стратегия ленивых конкретизаций

Суть ленивой конкретизации заключается в следующем. В качестве подстановки для неограниченной переменной выбирается не конкретный элемент эрбранова универсума, а *неопределенный эрбранов элемент* (НЭЭ), который, в дальнейшем, исходя из стратегии поиска необходимых термов для построения шага вывода, постепенно конкретизируется до основного терма, либо, в некоторых ситуациях, так и остается неконкретизированным. В данном случае необходимые термы обеспечивают возможность ответа на вопрос: НЭЭ постепенно конкретизируется таким образом, чтобы можно было на очередном шаге вывода построить новый ответ на какой–либо или заданный вопрос. Одновременно решается следующая задача: до какого терма конкретизировать НЭЭ, чтобы появился ответ на вопрос? Отметим, что НЭЭ первоначально появляется как часть ответа в одном вопросе, а его конкретизация производится при поиске ответов на другие вопросы. Под “постепенной конкретизацией” понимается процедура ленивых вычислений: НЭЭ конкретизируется настолько точно, насколько этого достаточно для ответа на текущий вопрос, т.е. конкретизация может быть неполной (т.е. не конкретизирующая НЭЭ до основного терма), например НЭЭ h конкретизируется до $f(h_1)$, где h_1 новый НЭЭ.

По своей природе НЭЭ схож с неконкретизированной универсальной пе-

ременной в том смысле, что он изменяем, т.е. потенциально конкретизируем, однако все такие изменения должны быть направлены только на конкретизацию НЭЭ, т.е. НЭЭ заменяется только на некий терм (возможно тоже содержащий НЭЭ), либо на другой НЭЭ, но не на переменную. Однако, при замене переменной на НЭЭ, НЭЭ обладает всеми свойствами основного терма.

Рассмотрим пример приложения этой техники в процессе построения логического вывода для следующей формулы.

$$\forall\colon \boldsymbol{True} - \exists\colon \boldsymbol{True} - \begin{cases} \forall x\colon \boldsymbol{True} - \exists\colon A(x) \\ \forall x\colon A(f(x)) - \exists\colon B(f(x)) \\ \forall\colon B(f(a)) - \exists\colon \boldsymbol{False} \end{cases}$$

На первом шаге вывода получен ответ $\{x \to h_1\}$ на первый вопрос, x является неограниченной переменной, h_1 — это неопределённый эрбранов элемент (НЭЭ). После первого шага атом $A(h_1)$ добавляется в базу. На втором шаге вывода получен ответ $\{x \to h_2\}$ на второй вопрос, и h_1 конкретизируется до $f(h_2)$, т.е. фактически ответом является подстановка $\{x \to h_2, h_1 \to f(h_2)\}$. После второго шага $B(f(h_2))$ добавляется в базу. Наконец, на третьем шаге получен тривиальный ответ на третий вопрос, и h_2 конкретизируется до a. Таким образом, можно проследить цепочку конкретизации НЭЭ: $\{x \to h_1\}, \{h_1 \to f(h_2)\}, \{h_2 \to a\}$, т.е. без использования НЭЭ, на первый вопрос необходимо было дать ответ $\{x \to f(a)\}$, прводящий в итоге к успешному опровержению формулы.

В общем случае существуют особые ситуации. Рассмотрим следующий пример.

Пример 3

$$\forall\colon \boldsymbol{True} \;-\; \exists\colon M(e) \begin{cases} \forall x, y\colon M(x) & - \;\; \exists\colon S(y), M(f(x)), T(x) \\ \forall x\colon T(x), S(e) & - \;\; \exists\colon Q(x) \\ \forall x\colon Q(x), S(f(e)) & - \;\; \exists\colon \mathbf{False} \end{cases}$$

Данная формула имеет вывод, например такой. Получаем ответ на первый вопрос подстановкой $\{x \to e, y \to e\}$, в результате в базу попадают факты $S(e), M(f(e)), T(e)$; ответ на второй вопрос есть $\{x \to e\}$, и в базу попадает $Q(e)$; полученных фактов в базе недостаточно для ответа на целевой вопрос, поэтому вновь отвечаем на первый вопрос, при этом возможно несколько вариантов ответа, но, что важно, переменную y необходимо заменить на $f(e)$. Выберем, например, следующий ответ $\{x \to e, y \to f(e)\}$ и в базу попадет факт $S(f(e))$ после чего на целевой вопрос ответ получается. Заметим, что на первый вопрос необходимо обязательно выбирать те подстановки, которые содержат $y \to e$ и $y \to f(e)$, они необходимы для ответа на второй и третий вопросы, соответственно. Формула устроена таким образом что первый и второй вопросы всегда имеют новые ответы.

Теперь рассмотрим работу стратегии ленивой конкретизации. Ответ на первый вопрос будет следующего вида $\{x \to e, y \to h_1\}$, где h_1 — НЭЭ, и в базу попадают следующие факты $S(h_1), M(f(e)), T(e)$. Ответ на второй вопрос — $\{x \to e, h_1 \to e\}$, в котором h_1 конкретизируется до e, и, соответственно, находящийся в базе факт $S(h_1)$ конкретизируется до $S(e)$. С этого момента начинается выполнение, фактически, циклической операции, поскольку целевой вопрос не имеет ответа: вновь получаем ответ на первый вопрос, и вновь в базу попадает факт $S(h_1)$, который при ответе на второй вопрос вновь конкретизируется до $S(e)$ и т.д. Однако, если пропустить ответ на второй вопрос, то при ответе на целевой, факт $S(h_1)$ конкретизируется до $S(f(e))$ и на этом вывод закончится.

В общем виде проблема заключается в том, что существуют ситуации, когда с формальной точки зрения НЭЭ конкретизируется корректно, но не там где это требуется, например, раньше, чем это необходимо при ответе на другой вопрос. Либо НЭЭ может конкретизироваться там, где это уже не требуется. Из подобных примеров видно, что ленивая конкретизация не всегда может использоваться напрямую. Необходимы дополнительные эвристические средства обеспечения логического вывода, учитывающие описанные только что проблемы.

Ограничение количества конкретизаций (ограничение 1). Два НЭЭ будем называть *подобными*, если они получены в результате выбора подстановки для одной и той же переменной. Такая ситуация имеет место если на один и тот же вопрос с неограниченными переменными произведено несколько ответов, и для неограниченных переменных все разы в качестве подстановки выступает новый НЭЭ.

Ограничение одинаковых конкретизаций для подобных НЭЭ позволяет исключить ситуации, когда НЭЭ конкретизируется там, где это уже не требуется, и, как следствие, появляется возможность конкретизировать его в другом месте. Для каждого НЭЭ хранится ссылка на переменную, для которой этот НЭЭ был выбран как подстановка. В вопросах каждой неограниченной переменной соответствует набор термов, до которых конкретизировались НЭЭ соответствующие данной переменной. Таким образом, имеется возможность отслеживать сколько раз и как конкретизировался данный НЭЭ. Если лимит конкретизаций исчерпан, то последующие конкретизации запрещены. Выбор конкретного числа, ограничивающего количество конкретизаций, определяется либо пользователем, исходя из его знаний о задаче, либо это число изначально устанавливается равным 1, и, далее, это число увеличивается в случае неуспешности вывода за определенное количество шагов или времени.

Использование данного подхода позволяет решить пример 3 за 5 шагов, если установить единице лимит конкретизаций для переменной y из первого вопроса. Поскольку h_1 не будет конкретизироваться до e второй раз, во втором вопросе.

Сохранение выражений, содержащих НЭЭ (ограничение 2). Представим другой вариант управления стратегией ленивых конкретизаций. Описанный выше вариант конкретизаций НЭЭ, в том числе используемый в примере 3, не сохраняет исходное выражение, содержащийся в котором НЭЭ конкретизируется. Отсюда возникает проблема потери информации. Например, если в примере 3 в базу попадает атом $S(h_1)$, то при необходимости конкретизации h_1, конкретизация производится не в самом атоме, а порождается новый атом, такой, как если бы был конкретизирован исходный, т.е. в случае рассматриваемого примера, к базе добавляется атом $S(e)$, при этом атом $S(h_1)$ сохраняется. НЭЭ h_1 сохранён, а значит, может использоваться в других вопросах.

Содержательно такой подход означает следующее. Можно считать, что для вопроса с открытыми переменными применяется вся совокупность возможных ответов, т.е. с использованием всех элементов эрбранова универсума. Понятно, что это технически нереализуемо из–за бесконечности эрбановского универсума, поэтому и используется техника ленивых вычислений. В идеале предполагается, что за один раз используются все возможные ответы на вопрос, но на самом деле используются только необходимые ответы. Например, если в некоторой формуле содержится вопрос $\forall x : True - \exists A(x)$, и $H^\infty = \{a, f(a), f(f(a)), ...\}$, то использование всех возможных ответов разом приводит к попаданию в базу элементов $\{A(a), A(f(a)), A(f(f(a))), ...\}$, которых бесконечно много. Вместо этого множеству ставится в соответствие элемент $A(h)$, где h — НЭЭ, и в дальнейшем этот элемент порождает необходимые элементы множества H^∞.

В данном случае можно заметить сходство со стратегией ленивых конкретизаций, без каких–либо ограничений. Преимущество подхода с сохранением выражений, содержащих НЭЭ, заключается в том, что выражение, содержащее НЭЭ всегда сохраняется, т.е. не может быть утрачено за счёт применения каких-либо подстановок.

Рассмотрим отдельно два случая. Если конкретизируемый НЭЭ содержится в подформуле-вопросе, и если конкретизируемый НЭЭ первоначально появился после ответа на вопрос с дизъюнктивным ветвлением.

В случае с дизъюнктивным ветвлением, вся совокупность ответов на вопрос означает не просто появление бесконечного конъюнкта, а бесконечно множества конъюнктов, поскольку база ещё и расщепляется. Т.е. конкретизация НЭЭ приводит не просто к порождению нового атома в конъюнкте, а к очередному расщеплению формулы, в той точке ЛВ, в которой впервые появился конкретизируемый НЭЭ. Данную точку легко отследить благодаря структуре ДСВ, в которой сохраняются все события произошедшие на каждом шаге вывода. Однако, ветвление целесообразно проводить не в этой точке, а во всех листах дерева, корнящегося из этой точки. Поскольку вставка ветвления в середину вывода потребует копирования для каждого нового узла той части вывода, которая была уже произведена. На рис. 2.9. схематично показано некоторое ДСВ. После конкретизации НЭЭ, получаем структуру

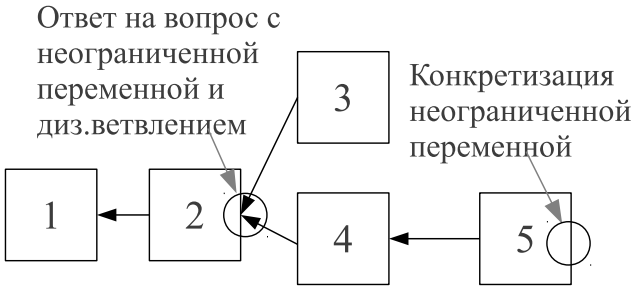

Рис. 2.9. Структура дерева состояний вывода до конкретизации

ДСВ, представленное на рис. 2.10.

В случае подформулы–вопроса, содержащего НЭЭ, конкретизация НЭЭ

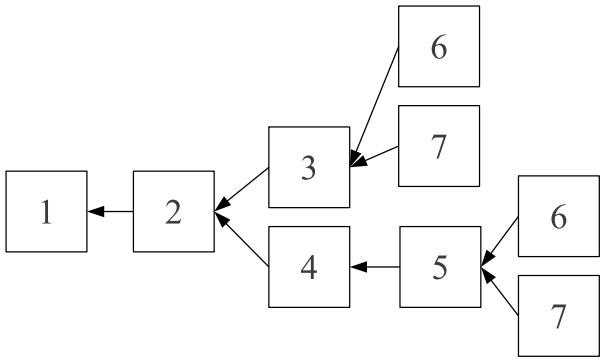

Рис. 2.10. Структура дерева состояний вывода после конкретизации

требует порождения новой подформулы–вопроса, отличающейся от исходной только в точке содержания НЭЭ. Возможно появление множества вопросов, усложняющее в целом структуру формулы и её вывод. Под усложнением структуры, в данном случае, понимается именно наращивание количества вопросов. Абсолютный размер формулы (в байтах) меняется незначительно, поскольку используются стратегии экономии памяти. Новую подформулу–вопрос целесообразно вставлять в той точке вывода (перед этим узлом), где первоначально добавился вопрос с НЭЭ. Предположим что на рис. 2.9. вопрос с НЭЭ впервые появляется в узле 2. Тогда если конкретизация НЭЭ потребуется в любом из последующих узлов, дерево преобразуется в вид, как на рис. 2.11. При этом новый узел 6 содержит только новый порождённый вопрос.

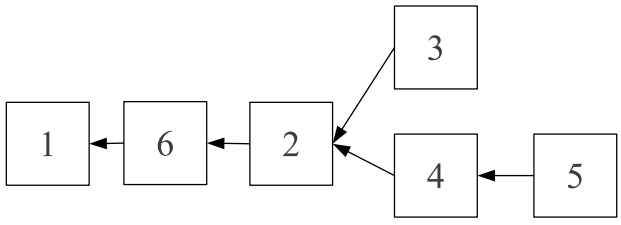

Рис. 2.11. Дерево состояний вывода после конкретизации в узле без дизъюнктивного ветвления

Ручное управление. Одним из вариантов решения проблемы является использование языка описания стратегий, с помощью которого можно, например, указать, что на второй вопрос необходимо ответить лишь один раз, либо использовать на первый вопрос сразу несколько подстановок содержащих $y \to h_1$ и $y \to h_2$. Это приведет к попаданию в базу двух фактов $S(h_1)$ и $S(h_2)$, один, из которых будет использоваться для второго вопроса, а другой для целевого. Такой вариант вполне приемлем, и может использоваться для решения задач. Такой подход в настоящее время пока не обеспечивает принципиальную выводимость, но для отдельных классов задач он уместен.

Отметим, что стратегии, основанные на использовании НЭЭ, создают некоторые проблемы совместимости с другими стратегиями. Во-первых, при использовании этой стратегии нарушается независимость базовых подформул, поскольку один и тот же НЭЭ может быть в разных подформулах: конкретизация общего НЭЭ в одной подформуле автоматически приводит к конкретизации его в другой подформуле. Этот факт, в частности, делает зависимыми процессы поиска ЛВ в базовых подформулах на параллельных вычислительных архитектурах.

2.4.2. Стратегия фильтрации эрбранова универсума

Другим вариантом (помимо стратегии ленивых конкретизаций) решения проблемы неограниченных переменных, является *стратегия фильтрации эрбранова универсума.* Данная стратегия позволяет избавиться от изложенных в начале главы проблем, но несколько расширяет пространство возможных ответов. Суть данной стратегии заключается в следующем. В консеквентах, атомы которых содержат неограниченные переменные вопросов, должны унифицироваться с одноимёнными атомами из всех других частей всей базовой подформулы, в противном случае атом из консеквента после попадания в базу не будет удовлетворять условию применения правила вывода (Опреде-

ление 3 Главы 1), поскольку никогда не выполнится процедура унификации при поиске ответов. Будем использовать данное свойство в стратегии. В качестве ответа для неограниченных переменных используются такие термы, что атом полученный подстановкой переменных на эти термы будет основным примером найденной унификации. Такой подход позволяет заранее сузить пространство эрбранова универсума, фильтруя заведомо бесполезные ответы.

Для ясности рассмотрим пример. Пусть имеется следующие вопросы ПО–формулы.

$$Q_1 = \forall x, y : -\exists : A(f(x, g(y)))$$

и

$$Q_2 = \forall x : A(f(e, x)) - \exists : False$$

В данном случае в консеквенте вопроса Q_1 содержится атом $A(f(x, g(y)))$. Одноимённый атом, это атом вопроса Q_2, а именно $A(f(e, x))$. Унификация двух данных атомов есть $A(f(e, g(y)))$. Основной пример данного атома есть любой, атом полученный из исходного заменой переменной y на любой элемент H^∞. Тогда ответ для вопроса $Q_1 = \{x \to e, y \to t\}$, где t любой элемент H^∞, например e. В базу попадёт факт $A(f(e, g(e)))$ и ответ на целевой вопрос будет $\{x \to e\}$.

Такой подход имеет сходство со стратегий ленивых конкретизаций, в которой конкретизация НЭЭ, по сути, означает сужение эрбранова универсума, до множества основных примеров конкретизации. В данном же случае ограничения на эрбранов универсум накладываются сразу, исходя из потенциальных вопросов. В случае стратегии ленивой конкретизации, конкретизация как раз производится при поиске ответов на эти потенциальные вопросы.

Данная стратегия позволяет всегда работать с полностью конкретизированной формулой, т.е. формулой не содержащей НЭЭ.

2.5. Стратегия k, m–ограничения

Данная стратегия формулируется следующим образом. Некоторый ответ применяется, если за последующие k шагов произойдет заданное событие, по меньшей мере m раз. Пользуясь терминологией ДСВ, это означает, что для данного узла дерева применяется ответ в случае, если построенное в результате дальнейшего вывода поддерево, коренящееся с этого узла, не превысит глубину k, и до этого момента произойдет m раз заданное событие. Данная стратегия хорошо совместима с ДСВ, и использует его возможности для возможных возвратов в выводе. Предложено три специальных случая данной стратегии.

k, m–опровержение. Заданный ответ выбирается в случае, если за последующие k шагов ЛВ будет опровергнуто как минимум m баз. Подобная стратегия, а именно k–опровержение, первоначально предложена А.К. Жерловым в [3] и реализована в системе КВАНТ/1 Черкашина Е.А. [20], где показана её состоятельность. В данной системе эта стратегия расширена вторым параметром m. Она позволяет сдерживать разрастание пространства поиска вывода, т.е., сдерживать излишнее ветвление ДСВ, что в некоторых случаях приводит к многократному усложнению вывода.

Отметим, что выбор параметров k и m, заранее не определён, и пользователю необходимо знать следующие нюансы. Во-первых, параметр k означает, что будут проверены все возможные ответы на глубине k шагов, т.е. произведен полный перебор, что в свою очередь может затратить большие ресурсы процессорного времени и памяти. Параметр m дополнительно усиливает, условие выбора исходного ответа. Поэтому, если заранее нельзя предположить какие выбрать эти параметры, то по умолчанию они устанавливаются $k = 1$ и $m = p$, где p количество непосредственных дочерних дизъюнктивных узлов вопроса, и в случае неудачи ЛВ, параметр k постепенно увеличивается,

а m уменьшается. Кроме того, если задать максимально возможное значение m, и не использовать других ограничений на полноту вывода, то можно производить полный перебор пространства поиска на глубину k.

k-неопровержимость. Для вопроса с дизъюнктивным ветвлением, какой-либо ответ принимается только в случае, если за k шагов не будет достигнуто опровержение базовой подформулы с использованием только вопросов без дизъюнктивного ветвления. Смысл такого подхода, как и в предыдущем варианте, заключается в том, чтобы сдерживать разрастание формулы из-за ответов на вопросы с дизъюнктивным ветвлением. Ответ для такого вопроса применяется только если нет иных (без ветвления) способов опровержения. Отметим, что в данном случае параметр m убран.

k, m–конкретизация. Это разновидность спецификации стратегии k, m–ограничения, она формулируется так. Ответ на вопрос принимается, если за последующие k шагов будет конкретизировано m НЭЭ. Эта стратегия также направлена на то, чтобы ограничить сложность представляемой формулы. С неконкретизированным НЭЭ ассоциируется много дополнительной информации и условий, описанных в разделе 2.4. о неограниченных переменных, что на уровне реализации влияет негативно. Поэтому чем больше и быстрее НЭЭ будут конкретизированы, тем лучше.

2.6. Параллельные стратегии

Повышение производительности достигается при помощи вышеописанных интенсивных методов, ориентированных на оптимизацию использования вычислительных ресурсов, а также при помощи экстенсивных методов, базирующихся на вовлечении в процесс дополнительных вычислительных ресурсов. В частности, это становится актуальным ввиду широкого распространения

многоядерных вычислительных систем общего назначения, в частности, рабочих станций.

Одним из популярных экстенсивных методов повышения производительности является разработка версий программ АДТ для кластерных архитектур в параллельном режиме исполнения ветвей подпрограмм. Такой подход реализован во многих современных системах АДТ, например в [52].

Рассмотрим методики и стратегии построения параллельных реализаций алгоритмов поиска ЛВ в исчислениях ПО–формул. Предложены следующие стратегии для построения параллельных схем алгоритмов поиска логического вывода.

Первая стратегия: опровержение баз. В случае, если вопрос имеет дизъюнктивное ветвление, то после ответа на этот вопрос, формула расщепляется и трансформируется в формулу с бóльшим количеством баз, т.е. в общем случае, количество базовых подформул увеличивается с каждым шагом вывода. С другой стороны, исходная формализация задачи в языке ПО–формул, опять же в общем случае, содержит более одной базовой подформулы. Для того, чтобы показать, что исходная формула противоречива, необходимо опровергнуть каждую из баз. Специфика исчисления ПО–формул позволяет производить логический вывод и опровержение этих баз независимо друг от друга. Данное свойство называется естественным ИЛИ–параллелизмом, который следует из того что в базах находятся лишь основные термы. Поэтому процедура опровержения каждой базы может выполняться в отдельном вычислительном процессе или на отдельном вычислительном устройстве.

Таким образом, первая стратегия, реализуемая в виде параллельной схемы алгоритмов, формулируется следующим образом: каждая базовая подформула, содержащая только основные термы в базе, опровергается независимо от других базовых подформул, а значит этот процесс выделяется в от-

дельный параллельный независимый от других процесс, синхронизируемый с другими процессами только на этапах его создания в момент расщепления формулы и завершения в момент установления выводимости/невыводимости. Во время жизни этого процесса к нему может поступать асинхронный сигнал завершения от супервизора, обозначающий, что выполнение процесса далее не имеет смысла, например, формула является неопровержимой, что было доказано в какой-либо другой ветке поиска доказательства, или пользователь остановил выполнение программы. Для программной реализации алгоритмов данной стратегии созданы подпрограммы жесткого копирования и маршаллинга/демаршаллинга, чтобы полностью скопировать базовую подформулу и обрабатывать её в отдельном процессе независимо, т.е. не разделяя оперативную память.

Вторая стратегия: поиск ответов на вопросы. Для применения каждого шага логического вывода, необходимо выполнять, в общем случае, поиск ответных подстановок для заданного вопроса. Поиск ответных подстановок не изменяет структуру формулы, и не использует общих изменяемых данных, это значит, что процессы поиска ответа на каждый вопрос независимы, а значит параллельны. Отметим, что поиск ответных подстановок для одного вопроса является намного менее трудоёмкой задачей, чем опровержение базовой подформулы, но тем не менее если конъюнкт вопроса содержит достаточно много атомов, то поиск всех ответных подстановок усложняется, за счёт увеличения пространства — в общем случае декартова произведения множеств подстановок для каждого атома из конъюнкта.

Третья стратегия: поиск подстановок для атомов вопроса. Теперь рассмотрим процедуру поиска подстановок для отдельно взятого вопроса. Как было сказано В Главе 1, подстановка θ является ответом, если выполняется условие $A\theta \subseteq B$, где B — конъюнкт вопроса, A — конъюнкт базы. Для

сохранения полноты необходимо хотя бы потенциально иметь в распоряжении все возможные ответы, из которых выбирается подстановка для данного шага. Ниже описана структура хранилища ответов. Наполнение каждого чанка хранилища подстановками производится параллельно, поскольку чанки независимы.

Свойства стратегий. Анализ, описанных выше стратегий, показывает, что они обладают свойством вложенности. Т.е., для того, чтобы опровергнуть одну базовую подформулу (первая стратегия) необходимо найти ответы на вопросы (вторая стратегия). В свою очередь для поиска ответа, необходимо найти подстановки для каждого атома из конъюнкта вопроса (третья стратегия).

Исходя из этого, данные стратегии можно разместить по степени эффективности (иерархия стратегий). Не трудно видеть, что время, затрачиваемое на опровержение базы, как минимум, не меньше, чем время, затрачиваемое на поиск ответных подстановок, а на практике, как правило, оказывается намного больше, так как для опровержения базы необходимо неоднократно ответить на некоторые вопросы. Аналогичные выводы делаются по отношению к другим стратегиям.

Кроме того, можно выделить единое для всех стратегий свойство —- свойство однородности (стратегии имеют единую структуру). А именно, все они, по сути, сводятся к применению некоторой операции (опровержение базы, поиск ответов и т.д.) для каждого элемента некоторого множества (базы, вопросы и т.д.).

Одной из рекомендаций при реализации описанных алгоритмов на кластерных вычислительных системах является правильное распределение задач между вычислительными узлами кластера, в зависимости от скорости коммуникации между ними. Например, программная реализация первой стратегии должна привязывать процесс к вычислительному узлу. Привязка процесса к

вычислительному узлу, в случае второй стратегии, возможна с увеличением конъюнкта вопроса и увеличением множеств подстановок, соответствующих каждому атому вопроса. В противном случае, этого не стоит делать, как и при реализации третьей стратегии, так как коммуникационные затраты, вполне вероятно, перекроют полезное время вычислений, и тем самым лишь ухудшат результат.

2.7. Равенства

Для обработки предиката равенства, как правило, крайне неэффективно напрямую использовать аксиомы равенства (рефлексивность, симметричность, транзитивность, подстановочность) без специальной адаптации алгоритмов АДТ. Например, если формула содержит лишь один бинарный функциональный символ f и один бинарный атомарный символ A, то в языке ПО–формул аксиомы равенства для такой формулы будут представлены следующим образом.

$$\begin{cases} \forall x\colon \boldsymbol{True} - \exists\colon x = x \\ \forall x_1, y_1, x_2, y_2\colon x_1 = y_1, x_2 = y_2 - \exists\colon f(x_1, y_1) = f(x_2, y_2) \\ \forall x_1, y_1, x_2, y_2\colon x_1 = y_1, x_2 = y_2, A(x_1, y_1) - \exists\colon A(x_2, y_2) \end{cases}$$

Как видно, для каждого функционального и атомарного символа из формулы ставится в соответствие подформула–вопрос — аксиома равенства. Явное использование таких аксиом, во-первых, усложняет структуру формулы (появляются лишние вопросы), во-вторых, значительно увеличивает число шагов вывода, в-третьих, генерирует много (потенциально бесконечно) фактов в базе, возможно не участвующих в выводе, а также мешающих выводу. Отметим, что данная проблема не так ужасна как в МР, поскольку в МР, в общем случае, порождаются дополнительные дизъюнкты, а это соответствует порождению дополнительных вопросов в ПО–формуле. В исчислении

ПО–формул порождаются лишь атомы-факты, за которыми проще наблюдать, в силу их более простой структуры, чем подформулы-вопросы (или дизъюнкты)

Очевидно, что вопросы–аксиомы равенства предназначены для того чтобы выводить те и только те атомы–факты, которые эквивалентны по модулю $E(B)$, другим атомам-фактам из базы B. В данном случае $E(B)$ это все равенства из базы B.

Для того чтобы не использовать на прямую аксиомы равенства, предлагается генерировать эквивалентные по модулю $E(B)$ атомы–факты с помощью систем переписывания термов (СПТ) [26].

Для построения СПТ по равенствам в базе, используется известный алгоритм Кнута–Бендикса (Knuth–Bendix completion procedure) [27]. Алгоритм Кнута–Бендикса, получая на входе множество атомов–равенств и редуцирующий порядок над термами [26] строит эквивалентную конвергентную СПТ. Конвергентность СПТ означает, что для каждого терма существует и только одна конечная форма (нормальная форма), которая может быть получена за конечное число переписываний. Под порядком над термами имеется ввиду отношение строгого частичного порядка над множеством термов. В нашем случае используется лексикографический порядок.

С помощью правил переписывания генерируются новые атомы–факты, эквивалентные, уже имеющимся в базе. В сочетании со стратегией удаления неиспользуемых фактов (стратегия экономии памяти), очевидно, что базу не будут добавляться ненужные сгенерированные факты.

Отметим, что такой подход не нарушает основных особенностей исчисления ПО–формул. Правило вывода сохраняется, остаётся единственным и унарным. Ответные подстановки по–прежнему зависят лишь от базы, и не зависят от других вопросов. Базы по–прежнему остаются независимы и содержат основные термы. Структура формулы не изменяется.

2.8. Выводы по главе

Использование представленных в даной главе методик привело к следующим результатам: значительная экономия памяти по сравнению с предыдущими версиями систем АДТ [20], влекущее за собой экономию вычислительных ресурсов за счёт того, что предотвращается излишнее копирование термов и целых подформул, производится удаление неиспользуемых фактов, и замедляется расзрастание формулы; рост производительности с использованием параллельных стратегий (при условии применимости этих стратегий); сокращение пространства поиска за счет стратегий k, m–ограничения, сдерживающих разрастание формулы из–за дизъюнктивного ветвления, и стратегий неограниченных переменных, позволяющих не перебирать эрбранов универсум; за счёт применения методик индексирования термов повышена эффективность доступа к выражениям ПО–формул, и как следствие повышена эффективность поиска ЛВ по времени; эффективный ЛВ с предикатом равенства, по сравнению с явным использованием аксиом равенства.

Глава 3. Реализация алгоритмов программной системы

Глава 3. посвящена реализации структур данных, стратегий, алгоритмов и программных средств разработки системы автоматического доказательства первопорядковых теорем в исчислениях позитивно–образованных формул. В качестве исходной информации для реализации выступают результаты, полученные в предыдущей главе диссертации. Реализация включает средства автоматизации анализа количественных и структурных характеристик процесса поиска ЛВ, а также средств управления ЛВ. В результате реализации создана программная система АДТ и среда поддержки разработки систем АДТ, ориентированные как на практические задачи, так и на абстрактные задачи, традиционно, считающиеся математическими.

3.1. Архитектура системы

На рис. 3.1. изображена архитектура программной системы. В основе подхода к проектированию архитектуры системы использовалась методика проектирования последовательной конкретизации (“сверху–вниз”) [10]. Архитектура представляет собой набор взаимодействующих друг с другом функциональных блоков (подсистем).

Программная система АДТ состоит из следующих функциональных блоков:

Супервизор — подсистема глобального управления процессом поиска ЛВ и реализации выбранных стратегий. Особенности некоторых стратегий требуют анализа и модификации имеющихся данных в оперативной памяти во всех подпрограммах поиска ЛВ. Для поддержки реализации стратегий реализована подсистема, которая анализирует характеристики процессов исполнения подпрограммами ЛВ и полученных в них результатов. По

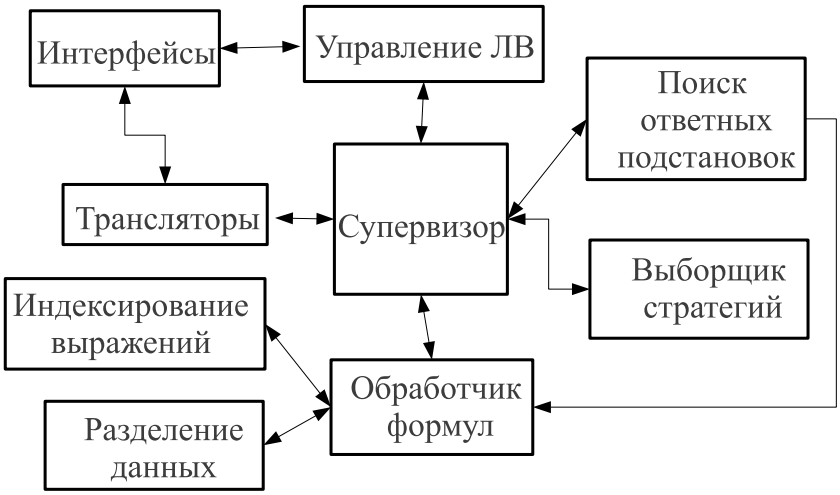

Рис. 3.1. Архитектура системы

запросам из подпрограмм Супервизор выдает необходимую статистическую информацию, на основе которой принимается то или иное решение в алгоритме реализации стратегии. Данная подсистема реализована как отдельный поток исполнения, ожидающий сигналов.

Выборщик стратегий — подсистема выбора стратегий, в которой на основе структурной и статистической информации о ПО–формуле и ЛВ выбирается та или иная подпрограмма, реализующая соответствующую стратегию поиска ЛВ.

Поиск ответных подстановок — подсистема поиска подстановок и ответных подстановок.

Обработчик формул — подсистема анализа и преобразований ПО–формул, в которой реализовано преобразование исходной ПО–формулы во внутреннее представление, снабжение этого представления вспомогательной информацией, используемой в ходе поиска ЛВ.

Транслятор — подсистема трансляции формул исчисления предикатов и КНФ, представленных в нескольких форматах, в ПО–представление.

Интерфейсы — подсистема интерфейсов к системе АДТ, включая интерфей-

сы прикладного программирования и интерфейс пользователя системы.

Разработанная система устанавливает не только тот факт, что формула является теоремой, но и в некоторых случаях позволяет распознавать некоторые противоречия. В частности, распознаются случаи, когда хотя бы одна из ветвей ДСВ не может больше изменяться по причинам, не связанным с ограничением ресурсов и при условии, что не использовалось ограничений, влияющих на полноту вывода. Такой случай характеризуется отсутствием допустимых вопросов и ответов на вопросы, т.е. ветвь принципиально неопровержима.

В следующих разделах опишем подробно каждую подсистему.

3.2. Супервизор

Супервизор управляет процессом поиска ЛВ и собирает статистику о процессах со всего ДСВ. Он обеспечивает алгоритмы информацией об ограничениях ресурсов, используемых эвристиках и т.п. Супервизор ссылается на все текущие листовые вершины ДСВ. Большинство стратегий реализовано на уровне супервизора, поскольку, в общем случае, необходимо использовать информацию о всей совокупности предшествующего или последующего вывода.

Наличие супервизора обусловлено тем, что некоторые частные события в процессе ЛВ могут повлиять на ЛВ в целом. Супервизор осуществляет наблюдение за поведением системы через Обработчик ПО-формул. Определены следующие операции: поиск выражений по ЛВ; слияние нескольких узлов ДСВ в один; выделение процедур опровержения базовой подформулы в отдельный независимый поток, с последующим ожиданием результатов опровержения; назначение стратегий; прерывание вывода и переход в режим интерактивного построения шагов; доступ к глобальному кэшу для использования кэшированных данных опровержения одной базовой подформулы в

опровержении другой; удаление листов дерева; удаление ветвей от листа до ближайшего ветвления; общий сбор потребляемых ресурсов, проверка ограничений; предоставление исходной информации о ПО–формуле.

3.3. Выборщик стратегий

Данная подсистема отвечает за выбор необходимых стратегий, улучшающих эффективность поиска ЛВ. Выбор осуществляется автоматически, исходя из структурных характеристик опровергаемой ПО–формулы. Ряд стратегий выбирается в самом начале поиска ЛВ и используется на протяжении всего процесса поиска ЛВ. Другие стратегии выбираются подпрограммами обработки текущего вопроса.

Ниже приведена таблица условий применения стратегий.

Таблица 3.1. – Выбор стратегий

Название стратегии	**Уровень применения**	**Условие применения**
k_1, m_1–опровержение	подформула-вопрос	есть дизъюнктивное ветвление
k_2, m_2–конкретизация	подформула-вопрос	есть неограниченные переменные
k–неопровержимость	подформула-вопрос	есть дизъюнктивное ветвление
Ленивая конкретизация (ограничение 1)	подформула-вопрос	пустой конъюнкт

Продолжение таблицы 3.1.

Название стратегии	**Уровень применения**	**Условие применения**
Ленивая конкретизация (ограничение 2)	подформула-вопрос	нет дизъюнктивного ветвления
Фильтрация эрбранова универсума	базовая подформула	Используется 1-ая параллельная стратегия или количество видов функциональных символов $< f_{max}$
Параллельная стратегия (вариант 1: по базам)	вся формула	есть вопрос с дизъюнктивным ветвлением
Параллельная стратегия (вариант 2: по вопросам)	базовая подформула	количество вопросов $> q_{min}$
Параллельная стратегия (вариант 3: по атомам вопроса)	подформула-вопрос, базовый конъюнкт	размер конъюнкта вопроса $> qc_{min}$ и размер базового конъюнкта $> bc_{min}$
Переписывание термов в базе	вся формула	наличие предиката равенства
Индексирование	вся формула	безусловное применение

Продолжение таблицы 3.1.

Название стратегии	**Уровень применения**	**Условие применения**
Мягкое разделение термов	вся формула	безусловное применение
Разделение данных по базам	вся формула	безуловно используется как часть ДСВ, но фактически проявляет себя в подформулах-вопросах с дизъюнктивным ветвлением
Агрессивное разделение термов	вся формула	любые ограничения памяти, установленные пользователем
Удаление неиспользуемых фактов	вся формула	достигнут предел памяти
Веса подформул	вся формула	любые ограничения памяти, установленные пользователем

Параметрам k, k_1, m_1, k_2, m_2 изначально присваивается значение 1. Дальнейшее их увеличение зависит от успешности применения стратегии с исходными параметрами. Если стратегия k, m–ограничения для всех ответов дала отрицательный ответ, то параметр k увеличивается. Когда достигнуто максимальное значение k, то изменяться параметр m. Максимальное значение k конфигурируется перед запуском программы.

Параметры $f_{max}, q_{min}, qc_{min}, bc_{min}$ конфигурируется перед началом работы программы. Подробнее о файле конфигурации ниже.

Файл конфигурации стратегий. Стратегии и критерии задаются при помощи конфигурационного файла. Формат файла представляет собой список двоек вида `<key>=<value>`, где ключ `<key>` — идентификатор стратегии, критерия и т.п., а `<value>` — значение данного ключа. Данные ассоциации помещаются в супервизор и, далее, используются в процессе поиска ЛВ.

Кроме того, в данном файле задаются следующие ограничения: максимальная глубина терма; максимальное количество вопросов; максимальная глубина вывода; максимальное количество базовых подформул; максимальный объём потребляемой памяти; ограничение по времени.

3.4. Поиск ответных подстановок

В данном разделе опишем механизмы поиска и хранения ответных подстановок.

3.4.1. Хранилище подстановок

Хранилище подстановок предназначено для эффективной организации доступа алгоритмов к ответными подстановками и возможности организации механизма бэктрэкинга. Хранилище позволяет осуществлять переходы в пространстве состояния поиска ЛВ, например, производить откат процесса на предыдущие состояния.

Каждому атому конъюнкта вопроса соответствует чанк возможных подстановок, обеспечивающих мэтчинг этого атома с атомами из базы. Использование чанков связано с тем, что необходимо точно определять на каком шаге и какие подстановки были найдены. Для поиска ответа на вопрос, необходимо использовать алгоритм композиции подстановок для каждого атома из вопроса. На рис. 3.2. представлена схема хранения подстановок для каждого атома.

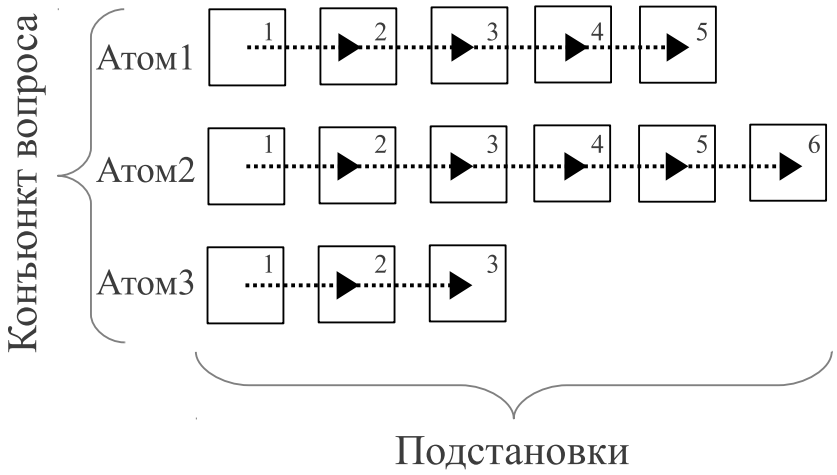

Рис. 3.2. Подстановки

Из этой структуры выделяется ответ на вопрос, конъюнкту которого соответствует свое хранилище. Для этого комбинируются подстановки из каждого чанка по одной. При этом подстановки должны быть совместимы. Две подстановки совместимы если их левые части равны, а правые унифицируемы. Например, если есть подстановки $\{x \to a\}$ и $\{x \to b\}$, где b есть константа, то эти подстановки несовместимы, поскольку a и b неунифицируемы. А подстановки $\{x \to f(a)\}$ и $\{x \to f(h)\}$, где h есть НЭЭ, совместимы, поскольку $f(a)$ и $f(h)$ унифицируемы с подстановкой $\{h \to a\}$. Результатом комбинации будет объединение всех совместимых и унифицирующих подстановок.

Перебор подстановок из чанков производится последовательно, это позволяет сохранить полноту.

3.4.2. Композиция подстановок

Хранилище подстановок содержит подстановки для отдельных атомов конъюнкта вопроса. Для того, чтобы получить ответ на вопрос, необходимо произвести композицию подстановок, соответствующих всем атомам конъюнкта вопроса. Хотя и существуют известные классические алгоритмы композиции подстановок, опишем такой, который учитывает особенности НЭЭ,

а также особенности реализации структуры обобщенного терма, а именно использование аргумента как места для подстановки.

Пусть дан список подстановок $QList$, дана пустая подстановка $Answer$, называемая результирующей. Алгоритм последовательно просматривает все подстановки $q \in QList$ и производит композицию $Answer$ и q. В случае удачи алгоритм переходит к следующему элементу $QList$, иначе алгоритм возвращает NULL, т.е. заканчивается неудачей.

Теперь опишем алгоритм композиции $Answer$ и q. Связью (binding) будем называть часть подстановки вида $\{x \to t\}$, т.е. одинарная подстановка для одной универсальной переменной или НЭЭ. Соответственно, подстановка есть список связей. Для каждой связи $b_i \in q$ произвести композицию b_i и $Answer$, в случае неудачи какого–либо шага, возвращается `NULL`.

Теперь опишем алгоритм композицию связи b и подстановки $Answer$. Для каждого элемента $a \in Answer$ проводим композицию b и a. В случае удачи, проводим композицию результата и $Answer$, берём следующий элемент $Answer$.

Рассмотрим вариант композиции двух связей b и a. Применить связь b. Если связь a стала рекурсивной (аналог проверки вхождения терма, occur check), т.е. правая часть содержит левую, то алгоритм заканчивается неудачей, связь b сбрасывается. Иначе, если левые части b и a различаются, то возвратить результат объединения b и a. Иначе, если левые части совпадают, то, если хотя бы одна из подстановок уже применена и правые части совпадают алгоритм возвращает любую из подстановок, иначе алгоритм терпит неудачу; если обе подстановки не применены, то выполняем алгоритм унификации правых частей. Если он закончился успехом, то возвратить подстановку и унификацию, иначе возвратить `NULL`.

3.4.3. Мэтчинг с НЭЭ

В системе реализован вариант классического алгоритма мэтчинга (matching), адаптированный для случая содержания в термах НЭЭ. Пусть дано два терма q и b, соответственно из базы и конъюнкта вопроса. Отметим, что поскольку b — это терм из базы, то он не содержит универсальных переменных. Задается пустая подстановка a, которая будет дополняться позже.

1. Если q — это константа. Тогда, если b тоже константа, проверить их корневые символы, если совпадают, то вернуть a, иначе вернуть `NULL`. Если b — это НЭЭ, то применить подстановку $\{b \to q\}$ и добавить её к a, возвратить a. Во всех остальных случаях возвратить `NULL`.

2. Если q — это экзистенциальная переменная. Если $b == q$, то возвратить a. Если b — это НЭЭ, то применить подстановку $\{b \to q\}$ и добавить её к a, возвратить a. Во всех остальных случаях возвратить `NULL`.

3. Если q — это универсальная переменная, то применить подстановку $\{q \to b\}$ и добавить её к a, возвратить a.

4. Если q — это НЭЭ. Если b тоже НЭЭ, то если $b == q$ вернуть a, иначе вывод разделяется на две части, в первом случае алгоритм возвращает `NULL`, во втором случае применить подстановку $\{q \to b\}$ и добавить её к a, возвратить a. Если b — тегирован как `FUNCTION`, то, если q является подтермом b возвратить `NULL`. Во всех остальных случаях применить подстановку $\{q \to b\}$ и добавить её к a, возвратить a.

5. Если q тегирован как `FUNCTION`. Если b тоже тегирован как `FUNCTION`, то если корневые символы q и b не совпадают, то возвратить NULL. Если совпадают, то это одноименные функции, одинаковой арности. Последовательно применить алгоритм мэтчинга для всех попарно соответствующих термов. Если применение алгоритма возвращает не `NULL`, то резуль-

тат применить (как подстановку) и добавить к a. Если получен `NULL`, то отменить a и вернуть `NULL`. Если b — это НЭЭ, то применить подстановку $\{b \to q_s(h_1, ..., h_n)\}$ и применить алгоритм мэтчинга ко всем парам (h_i, q_i), где q_s — это корневой символ s, n — арность q, h_i — новые НЭЭ, $i = \overline{1,n}$. Полученные результаты применить и добавить к a, возвратить a.

3.5. Обработчик формул

3.5.1. Копирование подформул

В системе реализовано два типа копирования подформул *мягкий* (soft copy) и *жесткий* (hard copy). Жесткий тип копирования предназначен для корректного копирования консеквентов и перемещения их к базе. Такое перемещение должно разорвать связь между переменными и их конкретизациями, для того чтобы освободить переменную для дальнейшего использования в других шагах вывода, но при этом, чтобы в базу попали именно те термы, до которых данная переменная конкретизирована. Кроме того, такой тип копирования нужен для параллельных стратегий, чтобы переместить формулу в полностью независимый процесс.

Второй тип копирования, мягкий, предназначен для корректной обработки НЭЭ с учетом бэктрэкинга. Поскольку при откатах поиска необходимо восстанавливать информацию о том какой НЭЭ был конкретизирован каким термом. Копирование НЭЭ осуществляется копированием его ссылки с сохранением его конкретизации. Потом такой НЭЭ в базе идентифицируется как конкретизированный и работа ведется с привязанным термом.

Копирование подформул в процессе шага вывода, требует, во–первых, разыменования всех экзистенциальных переменных, и всех неконкретизированных универсальных переменных. Поскольку структура копируемых под-

формул является древовидной, то для копирования применяется рекурсивный алгоритм, начинающий свою работу с корня дерева. Опишем подробно схему алгоритма копирования.

В начале работы алгоритма задан пустой словарь переменных, которые необходимо разыменовывать и соответствующих им разыменованных переменных. В скопированном узле (типового квантора) для его кванторных переменных создаётся новый список переменных той же длины что и исходный и того же типа (`EVARIABLE` или `AVARIABLE`), данный список есть копия исходного списка переменных, а в словарь заносятся соответствия между исходными переменными и новыми. Далее копирование конъюнкта требует создание нового конъюнкта такой же длины что и исходный, в который последовательно заносятся копии содержащихся в нём термов. Копирование термов производится рекурсивно. Создаётся новый корневой узел, в котором символ определяется следующим образом: если исходный символ это неконкретизированная переменная (либо `EVARIABLE`), и она уже входит в словарь, то её копией будет соответствующее ей значение в словаре. Это делается для того, чтобы одинаково разыменовать одинаковые переменные. Если переменная конкретизирована, то копируется значение функции `get_value()`. Отметим, что не существует ситуации, когда встречается неконкретизированная переменная, которой нет в словаре, поскольку используются только замкнутые формулы, а в словарь последовательно добавляются все переменные управляемые кванторами; если исходный символ имеет тип `CONSTANT`, `FUNCTION`, `ATOM` или `EVARIABLE`, не содержащаяся в словаре, то новый символ есть ссылка на исходный. Далее для терма копируются все его аргументы. После того как копирование конъюнкта завершено, копируется системная информация о типовом кванторе. Далее процедуре копирования дочерних узлов передаётся текущий словарь, который реализован с помощью чанков, для того, чтобы более глубокие (по сравнению с исходным) узлы могли разделять информацию о словаре переменных.

3.5.2. Шаг вывода

Когда выбран вопрос и ответ на этот вопрос, производится трансформация формулы, согласно определению 4. Данная трансформация состоит из следующих шагов:

1. Применить ответную подстановку (`answer.apply()`).
2. Произвести копирование подформул-консеквентов заданного вопроса с применением мягкого и жесткого копирования. Для каждого консеквента создаётся отдельный узел ДСВ.
3. Сбросить ответную подстановку.
4. Связывание полученных подформул–копий с имеющейся базовой подформулой. То есть, связывание полученных узлов ДСВ с соответствующим листом ДСВ.
5. Возврат результатов полученных в ходе данной операции.

3.5.3. Предобработка структуры формул

Предобработка структуры ПО–формул используется для упрощения формулы, которое, вероятно, влечет уменьшение сложности поиска ЛВ. Упрощение формулы достигается удалением очевидно ненужных фиктивных кванторов, слиянием нескольких вопросов в один, уменьшение глубины формулы и т.п. При этом происходит уменьшение сложности, в основном, сокращается длина вывода и пространство поиска переборных алгоритмов.

3.6. Выбор вопроса

Алгоритм выбора вопроса базируется на нескольких стратегиях. Первая стратегия, стратегия по умолчанию, задает вопросам приоритет в соответ-

ствии со следующим правилом. Сначала просматривается список целевых вопросов. Затем, если для них не получен ответ, то просматриваются простые вопросы, не являющиеся глубокими и не содержащие дизъюнктивное ветвление. Далее просматриваются глубокие вопросы, затем все остальные, в т.ч. с дизъюнктивным ветвлением.

Вторая стратегия реализует выбор вопроса по критерию. Критерий выбора — это некоторая количественная или структурная характеристика вопроса (подформулы). Примерами критериев выступают размер конъюнкта, наличие определенных термов в конъюнкте, частота удачного использования вопроса в ЛВ (рейтинг вопроса). Вопросам сопоставляется значение заданного критерия как приоритет использования вопроса. Вопросы, которым не присвоены значения критериев считаются низкоприоритетными.

3.7. Менеджер памяти

Как было сказано выше, одна из методик экономии памяти — это проектирование компактных структур данных. Под компактностью, в частности, понимается оптимизированное использование оперативной памяти. Выделение (allocation) оперативной памяти для структур (records) фиксированного размера реализовано на основе стандартного подхода, аналогичного [81] с формированием однонаправленного списка свободных структур (freelist). Массив структур отводится при помощи менеджера памяти операционной системы, и, затем, преобразуется в однонаправленный список свободных структур. Новые структуры выделяются из этого однонаправленного списка. При освобождении структура возвращается обратно в этот список для повторного использования. При исчерпании структур в списке и наличии свободной оперативной памяти выделяются новые массивы.

В системе АДТ для каждой разновидности структур (records) выделяется собственный массив, а также ведется собственный однонаправленный список

свободных структур. Сборка мусора синхронизирована со сборщиком мусора используемой системы программирования D.

3.8. Кэширование результатов

Учитывая множественность возможных ответов, возвратов поиска в выводе, большой глубины вывода, большого количества атомов в базе и т.д. целесообразно кешировать некоторые результаты чтобы не производить снова повторяющиеся вычисления.

Добавление уже имеющихся в базе атомов не допускается. Поэтому процедура поиска ответов работает всегда с новыми атомами, и производит всегда новые вычисления. Тем не менее полученные ответы в итоге могут совпадать. Все примененные ответы сохраняются, причем хранятся они как и база в чанках, расположенных по всему пути от листа до узла. Это позволяет делать корректные возвраты поиска с учетом сохранения информации о примененных ответах.

Некоторые подзадачи поиска ответных подстановок являются часто повторяемыми операциями. Поэтому для таких подзадач вводится кэширование результатов. Например, совмещение нескольких ответных подстановок.

3.9. Транслятор из TPTP в ПО–формулы

Одной из задач разработки системы АДТ является оценка производительности системы на стандартном наборе тестовых задач TPTP (Thousands of Problems for Theorem Provers) [93]. Этот набор задач включает в себя широкий спектр проблем, найденных авторами архива в литературе, а также задач, найденных пользователями и переданными в архив по e–mail.

Задачи в архиве представлены в виде текстовых файлов, состоящих из двух основных частей: комментария, где представлена метаинформация о

задаче, и собственно определения на специальном языке. Язык позволяет представлять формулы по отдельным подформулам, выносить подформулы в отдельный файл. Такое разделение позволяет собирать формулу из отдельных кусков, задавать библиотеки аксиом для отдельных теорий.

Сборка формулы из разных файлов осуществляется при помощи специальных утилит, например, `tptp4X`. Собранный файл, затем поступает в транслятор из TPTP–представления во входной язык системы. Транслятор реализован на основе утилиты `hotptp-yl-parser-verbose` [94], которая переводит входной TPTP–файл в дерево синтаксического разбора. Подсистема трансляции данной утилиты порождается на основе синтаксического анализа спецификации TPTP, поэтому всякий раз, когда происходит уточнение спецификации, утилита может быть обновлена автоматически (перепорождением и перекомпилированнием). Использование такого подхода позволяет оперативно адаптировать транслятор TPTP ко входному языку системы АДТ.

Дерево синтаксического разбора передается на вход подсистемы перевода, которая включает подсистемы импорта дерева разбора, преобразование дерева в формулу исчисления предикатов или КНФ, редукция формул по ряду правил, подготовка формулы ИП к преобразованию в ПО–формулу, собственно преобразование, редукция ПО–формулы, разыменование кванторных переменных, импликативное представление дизъюнктов, вывод в языке представления системы АДТ. Алгоритмы преобразования базируются на алгоритмах из [20]. В трансляторе также реализована подсистема перевода формул из входного языка [20], что позволяет также загружать некоторые задачи из [20].

3.10. Системные предикаты и вычисляемые термы

В логических языках программирования, например, Прологе [1], введены так называемые системные предикаты (встроенные предикаты, built-in

predicates), особенность которых заключается в том, что они или выполняют некоторое побочное действие, например, вывод на экран, чтение файла и др., или их истинность вычисляется из значений параметров, например, `var(X)` в Прологе определяет, является ли терм `X` переменной. Будем называть системным предикатом такой предикат, атомы которого входят в конъюнкты дерева ПО–формулы, но не участвуют непосредственно в ЛВ. Системные предикаты не имеют прямого отношения к формализации задачи, однако влияют на процесс ЛВ некоторыми побочными действиям, их истинностные значения вычисляются, выводят некоторую системную информацию. В данном разделе рассматривается использование системных предикатов для управления логическим выводом в процессе построения автоматического доказательства теорем в исчислении ПО–формул, то есть как способ задания дополнительных знаний о задаче в виде модификаторов стратегии, используемой по умолчанию.

В системе АДТ реализованы следующие системные предикаты:

`Next(L)` — переход к вопросу, помеченному идентификатором `L`. Предикат помещается в конъюнкт корневой вершины консеквента вопроса. Если на данный вопрос будет произведен ответ, то следующим вопросом, для которого будет производится выбор ответа, будет вопрос (множество вопросов), помеченный идентификатором `L`. Таким образом, при помощи данного предиката задаются варианты порядка ответа на вопрос.

`OffQuestion(L)/OnQuestion(L)` — отключение/включение вопроса с идентификатором `L`. Отключенный вопрос объявляется неактивным, то есть не принимает участие в логическом выводе, при этом в любой момент может быть заново включен.

`RemQuestion(L)` — удаление вопроса с идентификатором `L`.

`RemFact(L)` и `RemPatternFact(L)`. Удаление факта помеченного идентифи-

катором L, а также удаление всех основных примеров L, в случае если L — терм.

OffFact(L)/OnFact(L) — отключение и включение факта с именем L. Поведение подобно включению и отключению вопросов.

Write(T) — печать терма T.

Save(L) — пометить состояние процесса поиска ЛВ в данной базовой подформуле идентификатором L.

Rollback(L) — откатить (backtrack) состояние вывода в базовой подформуле до состояния L с утерей более поздних по отношению к L маркировок.

Commit(L) — фиксировать состояние базы L как неоткатываемое, при этом фиксируются все состояния, помеченные ранее, чем L .

Предикаты Commit и Rollback без параметров фиксируют или откатывают последнюю метку. Для пометки выражений используется следующий синтаксис E'(L), где E — выражение, L — метка. Кроме того, вводятся арифметические операции, используемые в конъюнкте вопроса и вычисляющие свои аргументы. Если арифметическая операция выполняется над полученными в подстановке аргументами, то подстановка используется в шаге вывода, иначе данная подстановка отвергается.

Рассмотрим некоторые ситуации, при которых перечисленные выше предикаты можно использовать и получать новые свойства процесса поиска ЛВ.

Ключевые точки в доказательстве. Нередко бывает так, что заранее известна некоторая точка, через которую должно пройти доказательство. Достаточно простым примером может служить задача поиска пути в городе между двумя точками, находящимися на разных берегах реки, при наличии одного моста. Понятно, что любой путь будет проходить через мост, однако

заранее неизвестно как именно строится этот путь. Предположим, что первая группа вопросов отвечает за правила движения в первой половине города, а вторая за движение во второй половине. Очевидно, что нет смысла пытаться отвечать на вопросы второй группы, пока не преодолен “мост”, а после его преодоления нет смысла отвечать на вопросы первой группы. Таким образом, перед началом доказательства, вопросы второй группы объявляются неактивными, а после ответа на специальный вопрос описывающий факт перехода через “мост”, неактивными объявляются вопросы первой группы, а вопросы второй группы включаются.

Очищение формулы. Другим типом задач являются задачи с некоторой дискретизацией шагов. Например, если моделируется переход системы из одного состояния в другое во времени, шаг такого перехода может быть эквивалентен нескольким шагам правила вывода. Часть выведенной информации, отвечающая за сам процесс перехода, может быть удалена, а другая часть, отвечающая за описание нового состояния, сохраняется. Примером может служить задача планирования движения робота в дискретном пространстве. Часть информации устаревает, и ее необходимо регулярно удалять из базы. Кроме того, устаревать могут и вопросы, отвечающие за конструктивные средства описания перехода из одного состояния в другое (движение ноги, кабины лифта и др.). Например, в базе должна оставаться история пройденного пути или решения поставленных роботу задач (обслужить все вызовы и т.п.).

Улучшение эффективности вывода. Рассмотрим простую задачу вычисления n–го элемента ряда Фибоначчи. На языке ПО–формул данная за-

дача вычисления 10–го элемента ряда формализуется следующим образом:

$$\forall\colon \boldsymbol{True} - \exists\colon f(1,1), f(2,2) - \begin{cases} \forall n, x, y\colon f(n,x), f(n+1,y) - \\ \qquad\qquad \exists\colon f(n+2, x+y) \\ \forall x\colon f(10,x) - \exists\colon \boldsymbol{False}, \end{cases}$$

где в процессе поиска подстановки θ операция «+» из значений ее аргументов вычисляется в константу (целое число), данная константа используется в θ вместо операции «+»; в консеквенте вопроса операция «+» заменяется на вычисленную константу во время применения подстановки.

В ходе ЛВ, база постепенно наполнится фактами и станет возможным ответ на целевой вопрос. При обычном выводе, с каждым ответом на первый вопрос в базу помещаются ответы, некоторые из которых будут дублироваться и как следствие могут производиться излишние вычисления. Вычисления ряда Фибоначчи подразумевает, что для вычисления последующего элемента достаточно использовать лишь два предыдущих элемента ряда. Это значит, что базу можно ограничить двумя последними добавленными элементами. Поскольку ответ на первый вопрос приводит к добавлению базы только одного нового факта, базу можно ограничить с помощью ввода системного предиката, удаляющего один самый старый элемент базы. Первый вопрос ПО–формулы с системным предикатом будет выглядеть так:

$$\forall n, x, y\colon f(n,x), f(n+1,y) - \exists\colon f(n+2, x+y)'(n+2), remFact(n)$$

при этом запись $'(L)$ означает «пометить терм меткой L». Соответственно база данной ПО–формулы выглядит так:

$$\exists\colon f(1,1)'1, f(2,2)'2$$

Удаление элементов из базы можно организовать без использования меток

атомов, если использовать RemPatternFact:

$$\forall n, x, y\colon f(n, x), f(n{+}1, y) {-} \exists\colon f(n{+}2, x{+}y)'(n{+}2), remPatternFact(f(n, \_))$$

Символ подчеркивания интерпретируется как и в Прологе в качестве анонимной переменной. Из базы будут удалены все факты, являющиеся основными примерами $f(n,_)$. Нетрудно заметить, что такой подход эквивалентен предыдущему, однако не требует дополнительных меток атомов.

Косвенное управление. Вывод (в смысле, ввод/вывод данных) некоторой системной информации косвенно можно отнести к управлению логическим выводом. Поскольку данная информация может интерпретироваться пользователем или возможно иной программой управления ЛВ, как некоторые входные данные для принятия решения. Среди выводимой информации отметим следующие: текущий шаг вывода, имя базы, имя вопроса, ответная подстановка, количество фактов в базе, количество опровергнутых баз, количество вопросов, количество потенциальных ответов, время ЛВ и печать некоторых термов.

Вычислимые термы. При решении прикладных задач может быть что заранее известна семантика формулы, область интерпретации и т.д. Отсюда некоторые сложные термы можно просто вычислять, а не выводить классическим образом Для этого каждому символу сигнатуры должна быть поставлена в соответствие некоторая функция.

Для обобщенного терма реализован метод `GTerm reduce()`, вычисляющий его значение. Вычисление допустимо, если всего его аргументы являются также вычислимыми термами, а все переменные и НЭЭ конкретизированы. В соответствии со строковым представлением символа ему задаётся семантика в виде лямбда выражения языка программирования D.

3.11. Среда реализации системы АДТ

Программная система АДТ реализована в среде программирования D [78, 23]. Язык D и его система программирования обладают рядом особенностей, позволяющих повысить продуктивность и упростить реализацию алгоритмов системы. Компилятор языка D генерирует машинный код по качеству сопоставимый с компиляторами языков C/C++, причем компилирование осуществляется в машинный код вычислительной системы, способный выполняться непосредственно микропроцессором без использования какой–либо виртуальной машины. Язык D обладает множеством полезных синтаксических элементов и возможностей, присущих языкам, построенным с использованием виртуальной машины. Так, D содержит в себе встроенные средства поддержки ассоциативных массивов. Ассоциативные массивы применяются для эффективной реализации некоторых структур данных, образующих основные классы системы АДТ. Они задают соответствие между символом (в терме) и адресом структуры в динамической оперативной памяти. Также D непосредственно поддерживает указатели (references), что позволяет эффективно получать доступ к некоторым структурами данных, обладающих ссылочной семантикой. Следует отметить, что стандартный комплект поставки D включает в себя мощную стандартную библиотеку, в которой есть средства для обработки строк. Среди стандартных средств программирования среды D необходимо выделить библиотеку поддержки параллельных вычислений, которая основана на принципах, заложенных в признанном лидере по параллельным вычислениям — языке Erlang [25]. Кроме того, одним из полезных свойств языка D является поддержка встроенного управления динамической оперативной памятью и сборщика мусора. Причем как процедурами выделения памяти, так и сборкой мусора можно управлять внешними библиотеками, т.е. настраивать управление оперативной памятью в системе АДТ. Эти возможности позволяют в рамках разработки системы АДТ настраивать (fine

tuning) процедуры обработки динамических структур данных на определенные задачи. Таким образом, среда программирования D представляет собой баланс между системным и прикладным уровнем программирования.

Программа АДТ состоит из следующих программных модулей:

prisnif.d — Головная программа и анализ входных параметров командной строки (218 строк исходного кода);

parser.d — Транслятор входного языка представления ПО–формул в структуры данных системы АДТ (640 строк исходного кода);

gterm.d — Программа для работы с обобщенными термами (1291 строк исходного кода);

proofnode.d — Узел дерева состояний вывода (458 строк исходного кода);

supervisor.d — Супервизор (902 строк исходного кода);

question.d — Программа работы с подформулами–вопросами, которые как особая часть формулы, выделена в отдельную структуру. Вопрос включает в себя информацию о типовом кванторе соответствующем вопросу и хранилище ответов для данного вопроса (550 строк исходного кода);

pchunk.d — Чанк (166 строк исходного кода);

qformulas.d — типовые кванторы (395 строк исходного кода);

symbol.d — символ (100 строк исходного кода);

answer.d — Программа работы с ответными подстановками, включающая в себя структуру связь `Binding` и собственно сам ответ `Answer` (200 строк исходного кода);

misc.d — разное (434 строк исходного кода);

pindex.d — программа индексирования термов и подформул путями (1200 строк исходного кода);

gtsharing.d — подсистема агрессивного разделения термов (804 строк исходного кода);

q_trans.pl — Подсистема трансляции формул с языка логики предикатов первого порядка и языка КНФ формата библиотеки TPTP, а также некоторая редукция и преобразования этих формул в ПО–представление (1055 строк исходного кода).

Всего реализовано 8413 строки кода.

Глава 4. Тестирование и сравнение программной системы

В данной главе продемонстрированы примеры применения разработанных инструментальных средств для решения практических задач.

4.1. Назначение программной системы

Разработанная система АДТ предназначена для поиска ЛВ в первопорядковом исчислении ПО–формул. Помимо установки того факта что формула является теоремой, имеются средства, которые в некоторых случаях устанавливают противоречивость исходной формулы (т.е. невыводимость её в исчислении ПО–формул). Входным языком представления формул может служить язык логики предикатов первого порядка, язык дизъюнктов и язык ПО–формул. Для первых двух языков взят формат библиотеки TPTP, для ПО–формул разработан формат близкий к используемому в системе КВАНТ Е.А. Черкашина [20].

4.2. Тестирование и сравнение

Тестирование системы проводилось на задачах библиотеки TPTP (Thousands of Problems for Theorem Provers, www.tptp.org). Данная библиотека является де–факто стандартом для тестирования систем АДТ. На момент написания работы библиотека содержала свыше 20000 задач, формализованных на языке предикатов первого порядка (FOF: first-order formula), конъюнктивной нормальной формы (CNF: clause normal form), типизированные формулы первого порядка(TFF: typed first-order form), типизированные формулы высшего порядка (THF: typed higher-order form). Кроме того, все зарегистрированные в библиотеке TPTP системы являются самыми передовыми

(“state-of-the-art systems”) современными системами АДТ, с которыми можно производить сравнение, на соответствие мировому уровню.

Все задачи классифицированы по следующим параметрам, а именно:

1. Рейтинг. Измеряется числом от 0.0 до 1.0. Задача с рейтингом 0.0 решается любой передовой системой АДТ, внесённой в библиотеку. Задача с рейтингом 1.0 ещё не была решена ни одной системой. Задачи с рейтингом от 0.04 уже относятся к классу сложных (“difficult”). Далее, говоря “сложная задача” будем иметь ввиду что задача обладает рейтингом от 0.04. Тестирование проведено по всем видам рейтинга от 0.0 до 1.0.

2. Предметная область задачи. Например, теоремы математического анализа (ANA), алгебры (ALG), геометрии (GEO) [58], головоломки (PUZ), медицины (MED), и др. Всего на момент тестирования имелось 48 областей.

3. Количественные характеристики формулы. Например, количество предикатов, функциональных символов, наличие предиката равенства, общий объем формулы.

4. Статус. Теорема (theorem), выполнима (satisfiable), невыполнима (unsatisfiable), неизвестно (unknown), открытая проблема (open), т.е. в принципе не решенная задача. Основное тестирование будем проводить по задачам статуса “теорма”.

5. Формализация. Язык логики предикатов первого порядка (FOF), конъюнктивная нормальная форма (CNF) и др.

Отметим, что библиотека регулярно обновляется и пополняется новыми задачами, требующих решения. Включенные в библиотеку системы, предназначены для решения задач из определенных областей, например, только для CNF и FOF, для теорем, для выполнимых задач и пр.

В данном раздле представлены примеры решения задач, описания и формулировки этих задач, формализации целей, сравнение с другими системами, зарегистрированными в TPTP, статистические данные о формулах. Даны комментарии по эффективности применяемых стратегий, проведены эксперементы с включением и отключением стратегий. Кроме того, даётся сводная информация о результатах тестирования и сравнения с другими система АДТ. С учетом того что решаются в основном первопорядковые теоремы статуса “теорема”, сравнение проводится с 31 системой АДТ, предназначеных для данных классов задач. Таблицы решенных задач по некоторым предметным областям представлены в приложении, кроме того в приложении представлены задачи, взятые с чемпионата мира среди систем АДТ 2012 (CASC-J6) [90].

4.2.1. Сводная информация о результатах тестирования

Решенные задачи по рейтингу представлены в таблице 4.1.

Таблица 4.1. – Рейтинг решенных задач

Рейтинг	**Всего задач**	**Решено**
Рейтинг 0.0–0.03	192	181
Рейтинг 0.04–0.20	435	373
Рейтинг 0.21–0.32	128	79
Рейтинг 0.33–0.49	115	44
Рейтинг 0.5–0.67	223	21
Рейтинг 0.68–0.92	72	6
Рейтинг 0.93–1.0	56	0

Лучшие показатели по предметным областям представлены в таблице 4.2.

Таблица 4.2. – Предметные области

Предметная область	**Всего задач**	**Решено**
Геометрия (GEO)	242	204
Менеджмент (MGT)	22	22
Синтаксические (SYN)	275	180
Семантический веб (SWB)	25	22

Самые сложные из решенных задач предсталены в таблице 4.3.

Таблица 4.3. – Самые сложные решенные задачи

Задача	**Рейтинг**	**Время**	**Шагов**
LCL652+1.015	0.92	32.1	185253
LCL656+1.020	0.92	1.24	845

Максимальный рейтинг задач, которые удалось решить 0.92, т.е. практически достигнута максимальная планка сложности проблем.

4.2.2. Примеры решенных задач

COM003+1. Формулировка: “The halting problem is undecidable” (“Неразрешимость проблемы остановки” [71]). Известная теорема о неразрешимости проблемы остановки машины Тьюринга: “Даны описание алгоритма и его начальные входные данные, требуется определить, сможет ли выполнение алгоритма с этими данными завершиться когда-либо”. Алан Тьюринг доказал, что не существует общего алгоритма для решения проблемы остановки, т.е. проблема остановки неразрешима.

Формализация данной задачи в языке логики предикатов первого порядка (FOF) включает 5 формул и 55 атомов. На сайте библиотеки TPTP:

www.tptp.org по номеру задачи можно увидеть её в формате FOF. Формализация её на языке ПО–формул представлена в приложении.

Рейтинг задачи: 0.33. Время решения: 0.005с. Количество шагов: 185.

Для сравнения приведем таблицу времени решения данной задачи одними из лучших, внесенными в TPTP, системами АДТ:

Таблица 4.4. – Результаты решения

Система АДТ	**Время решения (секунд)**
Vampire	0.01
iProver	0.47
leanCoP	29.25
EP	104.45
Zenon	0.08

Как видно, время решения задачи нашей системой лучше, чем время решения других систем АДТ. Добавим, что данная задача входила в список задач для чемпионата мира среди систем АДТ (CASC-J6), и из 16 систем АДТ 7 систем не смогли найти решение в рамках выделенного времени (5 минут).

При решении данной задачи, разработанной системой АДТ использовались стратегии: ленивая конкретизация (ограничение 1), $4,1$–опровержение, разделение данных по базе. Подключение параллельных стратегий, хотя и возможно, поскольку формула имеет дизъюнктивные ветвления, но в данном случае неоправданно, поскольку накладные затраты на создание процессов, перекрывают полезное время вычислений.

SYN548+1. Формулировка: $\Diamond\Box(\Box(p \vee \Box q) \Leftrightarrow \Box p \vee \Box q)$

Комментарий из библиотеки TPTP: Это проблема модальной логики [35],

транслированная в язык логики предикатов 1-ого порядка (FOF). Проблема легко решается системами АДТ для модальных логик, но трансляция этой задачи в язык логики предикатов 1-ого порядка является сложой задачей. Первопорядковая трансляция операторов $\Box$ и $\Diamond$ выглядит так: $[\Box P]x := (\forall y R(x,y) \rightarrow [P]y)$
$[\Diamond P]x := (\exists R(x,y) \& [P]y)$
$[P]x := P(x)$
P это атом, где мы стартуем с некоторым первоначальным контекстом o (первочанальный мир), т.е. мы преобразуем формулу P в $[P]o$.

Рейтинг 0.54. Время решения: 5.8с. Количество шагов: 82764.

Для сравнения приведем таблицу времени решения данной задачи одними из лучших, внесенными в TPTP, системами АДТ:

Таблица 4.5. – Результаты решения

Система АДТ	**Время решения (секунд)**
Vampire	0.02
iProver	1.71
EP	59.54
Zenon	12.07

Как видно, имеются системы, решившие задачу как значительно быстрее, так и значительно медленнее. Отметим, что рейтиг задач 0.54 говорит о том, что более половины систем АДТ не могут найти решение вообще.

При решении данной задачи использовались стратегии: ленивая конкретизация (ограничение 1), 8–неопровержимость, $8,1$–конкретизация, разделение данных по базе, агрессивное разделение термов.

LCL640+1.005. Формулировка: “In K, formula with A4 and Dum leading to Dum, size 5”. Описание проблемы дано в [28]. Представляет собой проблему модальной логики, выраженную в языке логики предикатов первого порядка.

Формализация данной задачи в языке логики предикатов первого порядка (FOF) включает 1 формулу и 121 атом. При этом структура формулы довольно сложная: глубина узлов достигает 16 уровней, имеется множество дизъюнктивных ветвлений.

Рейтинг задачи 0.62.

Время решения: 0.05с. Количество шагов: 821.

Для сравнения приведем таблицу времени решения данной задачи одними из лучших, внесенными в TPTP, системами АДТ:

Таблица 4.6. – Результаты решения

Система АДТ	**Время решения (секунд)**
Vampire	0.06
iProver	39.52
EP	174.46
Zenon	9.89

Как видно, время решения задачи нашей системой лучше, чем время решения других систем АДТ. Кроме того, отметим, что больше половины систем АДТ вообще не смогли решить эту задачу.

При решении данной задачи использовались стратегии: ленивая конкретизация (ограничение 1), $14,4$–опровержение, 4–неопровержимость, разделение данных по базе. Подключение параллельных стратегий, хотя и возможно, поскольку формула имеет дизъюнктивные ветвления, но в данном случае неоправданно, поскольку накладные затраты на создание процессов, пере-

крывают полезное время вычислений.

MSC014+1. Данная задача интересна тем, что не является теоремой. Тем не менее, с использованием полной стратегии вывода (т.е. без использования ограничивающих стратегий), данный факт устанавливается, в силу запирания ДСВ.

Рейтинг 0.33. Время решения 0.0007с. Другие системы, устанавливают данный факт тоже со временем близким к 0.

4.2.3. Крупные задачи.

Рассмотрим поведение системы при решении некоторых крупных задач, формализация которых превышает 100 Кб. Как правило, это задачи с формализацией очень большого количества аксиом, многие из которых не пригождаются в выводе, т.е. ответы на многие вопросы заведомо не приносят никакой пользы, более того, наоборот, захламляют вывод. Основной тактикой при решении таких задач является, во-первых, повсеместное использование экономии памяти, тщательное использование собранной статистики для удаления ненужных фактов и вопросов, и, во-вторых, использование знаний о задаче, для выявления заведомо плохих ветвей вывода.

CSR025+3. Рейтинг 0.17. Время решения 1с. Для сравнения, Vampire 0.14с, EP 0.7с, iProver 1.5с. Данная задача содержит 1 группу аксиом, включающую 8006 формул и 13036 атомов.

CSR064+3. Рейтинг 0.58. Время решения 4.4с. Для сравнения, EP 0.24с, iProver 5.8с. Количество формул 10189. Количество атомов 10803.

4.2.4. Сравнение с системами АДТ ИДСТУ СО РАН

Задача “о паровом катке”, как демонстрация решения сложной задачи была представлена в диссертации Черкашина Е.А. [20]. В библиотеке TPTP на данный момент она имеет три формализации PUZ031+1 (рейтинг 0.08), PUZ031+2 (рейтинг 0.04), PUZ031+3 (рейтинг 0.08), и попрежнему входит в число сложных задач.

Решение всех трёх вариантов осуществляется за 0.005 (близко к 0), 0.002 (близко к 0) и 0.4 секунды соответственно. Глубина вывода 19 (является минимально возможным выводом). Для сравнения, EP, Vampire, Otter, время близко к 0.01.

То есть, заметно значительное повышение производительности, как с точки зрения количества шагов, так и с точки зрения затраченного времени.

4.2.5. Комментарии по стратегиям

Ленивая конкретизация. Эффективность данной стратегии по сравнению с прямым перебором эрбранова универса, даже не поддается сравнению, поскольку второй вариант как правило не приводит вообще к решению задачи за приемлемое время. Наиболее важную роль, в данной стратегии, играют ограничения, позволяющие значительно сокращать число шагов вывода по сравнению с использованием данной стратегии без ограничений. Сокращение числа шагов вывода, в данном случае, влечет и сокращение времени решения. Наилучшим вариантом использования ограничений является применение ограничения 1 в случае вопросов с дизъюнктивным ветвлением, и ограничения 2 в случае вопросов без дизъюнктивного ветвлеия. Неформально, по степени важности данный подход мы ставим на первое место.

Фильтрация эрбранова универса. Данный подход лучше всего подходит при совмещении работы с неограниченным переменным и 1-ой параллель-

ной стратегии, для того что бы избежать появления зависимых баз, из-за попадания туда НЭЭ, которые изменяются в процессе поиска ЛВ. Кроме того, эффективность поиска ЛВ, с использованием данной стратегии, не снижается по сравнению с ленивой конкретизацией в случае если формула содержит сравнительно мало функциональных символов (обычно 0–1).

Экономия памяти. Использование ДСВ, позволило реализовать многие другие стратегии и одновременно с этим экономить память разделяя общие данные ветвящихся формул. Агрессивное разделение данных и удаление излишек, позволило применить систему для решения некоторых задач с крупной формализацией, а стратегия выбора ответа приводящего к формуле с наименьшим весом, позволило максимально оттянуть момент достяжения лимита выделенной памяти. Для некоторых задач, например SYN548+1, указанной выше, снятие запрета на использование повторов подстановок, приводит к десятикратному повышению времени решения задач.

k, m-ограничения. Эффективность данной стратегии, проявляется в задачах с дизъюнктвиынм ветвлением. Во всех случаях уместного использования данной стратгии и правильно выбранных параметров, достигает заметное повышение эффективности. Кроме того, данный подход позволяет реализовать полный перебор пространства поиска, что позволило найти для некоторых задач минимальный вывод.

Параллельные стратегии. Важным свойством параллельных схем алгоритмов является их масштабируемость, т.е. степень повышения эффективности с увеличением количества вычислительных элементов (ВЭ). Поэтому основной характеристикой является не конкретное время исполнения программ, а соотношение времени исполнения программы в параллельном режиме на заданном количестве ВЭ к времени исполнения этой же программы

на одном ВЭ при различном количестве ВЭ.

Эксперименты проводились на задачах, формализация которых в языке ПО–формул обладает необходимыми свойствами для испытания параллельных стратегий, а именно: дизъюнктивное ветвление, большое количество вопросов, крупные конъюнкты вопроса. Результаты находятся в соответствии с представленной иерархией стратегий. На рис. 4.1. представлены результаты данного тестирования.

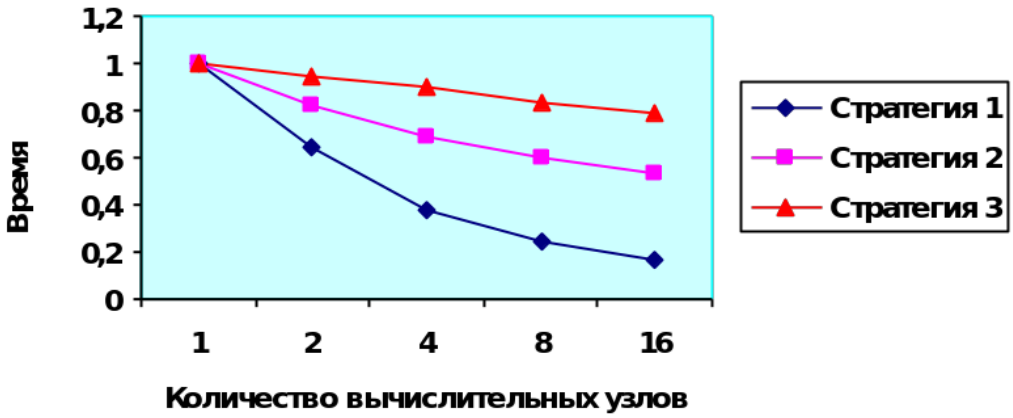

Рис. 4.1. Результаты тестирования параллельных стратегий

Наибольшую эффективность, как и следовало ожидать, показала первая стратегия, естественно при наличии дизъюнктивного ветвления в исходной формуле. Под эффективностью понимается уменьшение затрачиваемого времени с увеличением количества вычислительных узлов.

Равенства. Стратегия генерации эквивалентных атомов в базе, хотя и даёт принципиальное опровержение формулы, и повышение эффективности по сравнению с прямым использованием аксиом равенства, тем не менее, как правило, проигрывает другим системам АДТ при решении задач с равенствами. Однако, использование такого подхода, обусловлено желанием сохранить основные положительные свойства ПО–формул.

Менеджер памяти и кэширование. Помимо использования стратегий из главы 2., значительные результаты в повышении эффективности поиска ЛВ достигнуты благодаря использованию менеджера памяти и кэширования результатов.

4.3. Выводы по главе

Система протестирована на разных классах задач, в основном взятых из библиотеки TPTP.

Тестирование решенных задач, подтвердило предположение авторов исчисления ПО–формул, о том что крупноблочная структура формул и правила вывода позволяет повысить эффективность поиска ЛВ. Задачи, формализация которых в языке ПО–формул обладает свойством крупноблочности, а именно больших конъюнктов, решаются более эффективно, в сравнении с другими системами АДТ. Кроме того, наблюдается небольшой выигрыш в задачах, формализация которых в языке ПО–формул, включает экзистенциальные переменные и глубокие вопросы. Все перечисленные свойства: крупноблочность, явное использование двух видов кванторов, сохранение исходной структуры знания, являются основными отличительными свойствами от резолютивных методов. Продемонстрирована эффективность данных отличий.

С точки зрения предметных областей, наилучшая эффективность системы показана на задачах геометрии, менеджмента, семантического веба, и некоторых других.

Рейтинг решенных задач варьируется от 0.0 до 0.92 (с учетом того что сложные задачи начинаются от рейтинга 0.04), т.е. решаются, в том числе сложнейшие задачи библиотеки TPTP, в том числе многие из них более эффективно чем другими лучшими системами АДТ. Выигрыши по сравнению с ними выражается меньшим временем или количеством шагов вывода. Таким

образом, делается вывод, что разработанная система соответствует мировому уровню.

Манипуляцию с включением и отключением стратегий наглядно показывают, что они положительно влияют на эффективность поиска ЛВ. То есть, подтверждено на практике положительное влияние предложенных стратегий повышения эффективности поиска ЛВ.

Ещё одним результатом является значительное расширение класса решаемых задач по сравнению с предыдущими системами АДТ базирующимися на исчислении ПО–формул. Для раннее решаемых задач не только повышена эффективность их решения, но и достигнуты некоторые пределы эффективности, в частности найден минимальный вывод некоторых задач.

Лучшие системы, включенные в библиотеку TPTP имеют приоритет при решении задач содержащих много предикатов равенства.

Заключение

В диссертации представлены результаты исследования исчисления ПО–формул, как формализма для автоматического доказательства теорем (АДТ). Основным результатом является программная система АДТ, в которой реализован ряд стратегий повышающих эффективность поиска логического вывода (ЛВ). При этом эффективность формализована по следующим критериям: время, затрачиваемое на поиск ЛВ; количество шагов полученного ЛВ, количество потребляемой памяти в процессе поиска ЛВ; ширина классов решаемых задач программной системой АДТ.

Среди предложенных и реализованных стратегий имеются как адаптированные из других методов АДТ для исчисления ПО–формул, так и оригинальные.

Применение системы для решения ряда задач из библиотеки TPTP показало, что система соответствует мировому уровню в данной области. Указаны классы задач, как с точки зрения формы их представления, так с точки зрения предметных областей, на которых разработанная система выигрывает у основных мировых лидеров.

В рамках диссертации получены и на защиту выносятся следующие результаты:

1. Разработаны новые программные методики и технологии эффективного поиска ЛВ в исчислении ПО–формул: специальные структуры представления ПО–формул, стратегии поддержки неограниченных переменных, k, m–ограничение, алгоритмическая поддержка предиката равенства без явного использования аксиом равенства. Разработанные методики реализуют полезные свойства исчислений ПО–формул в программных системах АДТ.

2. Успешно адаптированы и применены существующие методики повыше-

ния эффективности поиска ЛВ для исчисления ПО–формул: индексирование термов, параллельные схемы алгоритмов, разделение данных для термов. Адаптация сохраняет полезные свойства исчислений ПО–формул.

3. Реализована новая версия программной системы для эффективного поиска ЛВ в исчислении ПО–формул первого порядка с функциональными символами и предикатом равенства. Создан программный инструментарий и технологии разработки специализированных версий программы АДТ, направленные на решение определенных классов задач АДТ.

4. Разработана инфраструктура тестирования алгоритмов и стратегий ЛВ для созданной программной системы АДТ на задачах из библиотеки TPTP. Выделены классы задач, которые система решает эффективнее, чем другие передовые системы АДТ. Значительно расширен класс успешно решаемых задач по сравнению с предыдущими системами АДТ, базирующимися на исчислении ПО–формул.

Таким образом поставленная цель диссертационной работы достигнута. Необходимые для достижения цели задачи решены.

Из особенностей, препятствующих внедрению разработанных инструментальных средств, на наш взгляд, являются:

1. Условие достаточно хорошего понимания принципов логического программирования и особенностей формализма ПО–формул у пользователя;

2. Проблема равенств решена не так эффективно, как в иных системах АДТ.

Проблема 1 может быть частично или полностью решена путём развития пользовательских интерфейсов и путем просвещения пользователей. Проблема 2 требует дальнейшего отдельного исследования.

Перспективы развития естественным образом делятся на два класса: развитие фундаментальной части языка и исчисления ПО–формул, в частности исследования вопросов построения модальных исчислений, дескриптивных, высшего порядка, использования табличного и обратного методов; и непосредственно расширение класса решаемых задач, в частности применение в области семантического веба, биоинформатикии, верификации, открытых математических проблем.

Литература

1. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG. – М.: Издательский дом “Вильямс”, 2004. – 640с.

2. Бутаков М.И., Курганский В.И. О решении задачи трансляции планированием вычислений на основе позитивно–образованных формул // Системы управления и информационные технологии, 2007. – С. 120–123.

3. Васильев С.Н., Жерлов А.К., Федунов Е.А., Федосов Б.Е. Интеллектное управление динамическими системами. – М.:Физматлит, 2000. – 352с.

4. Васильев С.Н. Об исчислениях типово-кванторных формул / Васильев С.Н., Жерлов А.К. // ДАН, Т. 343, N 5, 1995, С. 583–585.

5. Гильберт Д., Аккерман В. Основы теоретической логики. – Государственное издательство иностранной литературы, 1947. – 153с.

6. Гулямов Ш.Б. Математическое и программное обеспечение решения первопорядковых логических уравнений. канд. дисс. текст. ИДСТУ СО РАН, Иркутск, 1997. – 155с.

7. Давыдов А.В. Исчисление позитивно-образованных формул с функциональными символами // Прикладные алгоритмы в дискретном анализе: сб. науч. тр. /Под ред. Д-ра физ.-мат. Наук, проф. Ю.Д. Королькова. – Иркутск : Изд-во Иркут. гос. Ун-та, 2008. – 157 с. – (Дискретный анализ и информатика, Вып. 2). – С. 23–49.

8. Давыдов А.В., Ларионов А.А. Об исчислении позитивно-образованных формул для автоматического доказательства теорем. / Тр. 5-го международного симпозиума по компьютерным наукам в России. Семинар “Семантика, спецификация и верификация программ: теория и приложения”. 14-15 июня 2010, Казань. – С. 109–116.

9. Давыдов А.В., Ларионов А.А., Черкашин Е.А. Об исчислении позитивно-образованных формул для автоматического доказательства теорем. // Моделирование и анализ информационных систем. 2010.Т. 17, N 4. – С. 60–69.

10. Йодан Э. Структурное проектирование и конструирование программ. – М.:Мир, 1979. – 415с.

11. Ларионов А.А., Терехин И.Н., Черкашин Е.А., Давыдов А.В. Программная система КВАНТ/4 для автоматического доказательства теорем // Труды ИМЭИ ИГУ. Математика и информатика : сб.научных трудов / под ред.: Ю. Д. Корольков [и др.]. - Иркутск : Изд-во ИГУ, 2011. - Выпуск 1. – С. 77–85.

12. Ларионов А.А., Черкашин Е.А. Параллельные схемы алгоритмов автоматического доказательства теорем в исчислении позитивно-образованных формул. // Дистанционное и виртуальное обучение. Февраль 2012. No. 2. – С. 93–100.

13. Ларионов А.А., Черкашин Е.А., Давыдов А.В. Программная система для автоматического доказательства теорем в исчислении позитивно-образованных формул. / Винеровские чтения / Труды IV Всероссийской конференции. Часть II. - Иркутск: ИрГТУ, 2011. – С. 190–197.

14. Ларионов А.А., Черкашин Е.А., Терехин И.Н. Системные предикаты для управления логическим выводом в системе автоматического доказательства теорем для исчисления позитивно-образованых формул. // Вестник Бурятского государственного университета, 2011, выпуск 9. Серия Математика. Информатика. – С. 94–98.

15. Макаров И.М., и др. Логика и компьютер. Выпуск 5. – М.:Наука, 2004. – 207с.

16. Мендельсон Э. Введение в математическую логику. – М.: Книжный дом “ЛИБРОКОМ”, 2010. – 320с.

17. Маслов С. Ю. Обратный метод установления выводимости в классическом исчислении предикатов. / Маслов С. Ю. // ДАН СССР, Т. 159, N. 1. 1964. – С. 17–20.

18. Непейвода Н.Н. Прикладная логика. – Новосибирск: Издательство Новосибирского Университета, 2000. – 521с.

19. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. – М.:Наука, 1983. – 360с.

20. Черкашин Е.А. Программная система КВАНТ/1 для автоматического доказательства теорем. канд. дисс. текст. / Черкашин Е.А. / ИДСТУ СО РАН, Иркутск, 1999. – 155с.

21. Черкашин Е.А. Разделяемые структуры данных в системе автоматического доказательства теорем КВАНТ/3. / Черкашин Е.А. // Вычислительные технологии. 2008. Т. 13. – С. 102–107.

22. Чёрч А. Введение в математическую логику: Том 1. – М:. Книжный дом “ЛИБРОКОМ”, 2009. – 480с.

23. Alexandrescu A. The D Programming Language. – Addison-Wesley Professional, 2010. – 460p.

24. Aravindan C., Baumgartner P. Theorem Proving Techniques for View Deletion in Databases // Journal of Symbolic Computation - Special issue on advances in first-order theorem proving. Volume 29 Issue 2, Feb. 2000. pp. 119–147.

25. Armstrong J. Programming Erlang. – The Pragmatic Programmers, 2007. – 519p.

26. Baader F., Nipkow T. Term Rewriting and All That. – Cambridge University Press, Cambridge, 1988. – 316p.

27. Bachmair L., Dershowitz N., Plaisted D. A. Completion without failure. // In Resolution of Equations in Algebraic Structures, Vol. 2: Rewriting Techniques, 1989. pp. 1–30

28. Balsiger P., Heuerding A., Schwendimann S. A Benchmark Method for the Propositional Modal Logics K, KT, S4 // Journal of Automated Reasoning Volume 24, Number 3, 2000. pp 297–317.

29. Beth E. W., Semantic entailment and formal derivability / Mededlingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde, N.R. Vol 18, no 13, 1955, pp 309–42. Reprinted in Jaakko Intikka (ed.) The Philosophy of Mathematics, Oxford University Press, 1969.

30. Bibel W., Korn D., Kreitz C., Schmitt S. Problem-Oriented Applications of Automated Theorem Proving // Design and Implementation of Symbolic Computation Systems. International Symposium, DISCO '96 Karlsruhe, Germany, September 18–20, 1996. pp. 1—21.

31. Blackburn P. Automated Theorem Proving for Natural Language Processing

32. Bondyfalat D., Mourrain B., Papadopoulo T. An Application of Automatic Theorem Proving in Computer vision // Automated Deduction in Geometry. Second International Workshop, ADG'98 Beijing, China, August 1–3, 1998. pp. 207–232.

33. Bourbaki N.. Theory of Sets. – Springer, 2nd Revised edition, 2004. – 424p.

34. Brak R.L., Fleuriot, J.D., McGinnis, J. Theorem proving for protocol languages // In: The Second European Workshop on Multi-Agent Systems, Barcelona, Spain. 2004.

35. Chellas B.F. Modal Logic: An Introduction / Cambridge University Press. February 29, 1980. – 312p.

36. Claessen K., Hähnle R., Mårtensson J. Verification of Hardware Systems with First-Order Logic //

37. D'Agostino M., Gabbay Dov M., Hähnle R., Posegga J., eds. Handbook of Tableau Methods / Kluwer. Academic Publishers, Dordrecht 1999. – 680p.

38. Darvas Á., Hähnle R., Sands D. A Theorem Proving Approach to Analysis of Secure Information Flow // SPC'05 Proceedings of the Second international conference on Security in Pervasive Computing. pp. 193–209.

39. Davydov A.V., Larionov A.A., Cherkashin E.A. On the calculus of positively constructed formulas for automated theorem proving // Automatic Control and Computer Sciences (AC&CS). 2011. Volume 45, Issue 7, pp. 402–407. [Об исчислении позитивно-образованных формул для автоматического доказательства теорем]

40. Degtyarev A., Voronkov A. The Inverse Method. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning. – MIT Press, Cambridge, 2001. – pp. 181–270.

41. Frege G. Begriffsschrift: eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle, 1879.

42. Garcia-Molina H., Ullman J. D., Widom J. Database Systems: The Complete Book / Prentice Hall. 2009. – 1210p.

43. Untersuchungen uber das Logische Schliessen / Mathematische Zeitschrift 39. – pp. 176–209, 405–431.

44. Godel K. Die Vollstandigkeit der Axiome des Logischen Funktionenkalkuls. - Monatsh. Math. Phys., 1930, 37. – pp. 349–360.

45. Godel K. Uber formal unentscheidbare Satze der Principia Mathematica und verwandfer Systeme, I. - Monatsh. Math. Phys., 1931, 38. – pp. 173–198.

46. Graf P. Substitution Tree Indexing // Proceedings of the 6th International Conference on Rewriting Techniques and Applications. 1995. pp. 117–131.

47. Graf P. Term Indexing (Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence). – Springer, 1 edition. March 27, 1996.

48. Heijenoort J. From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931, 1967 (1999).

49. Hommersom A.J., Lucas P.J.F., Bommel P. Automated Theorem Proving for Quality-checking Medical Guidelines // CADE-20 : 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005 : Workshop on Empirically Successful Classical Automated Reasoning (ESCAR).

50. Kortenkamp U., Richter-Gebert J. Using Automatic Theorem Proving to Improve the Usability of Geometry Software // In Proceedings of the Mathematical User Interfaces Workshop, 2004.

51. Larionov A.A., Cherkashin E.A., Davydov A.V. Theorem Proving Software, Based on Method of Positively-Constructed Formulae // MIPRO 2011. 34-th international convention on information and communication technology, electronics and microelectronics. Vol. III. May 23-27, 2011. Croatia, Opatija. //MIPRO: Croatia, 2011. pp. 365–368. [Программа для автоматического доказательста теорем, основанная на исчислении позитивно-образованных формул]

52. Letz R., Stenz G., Wolf A. P-SETHEO: Strategy Parallel Automated Theorem Proving // Proceedings of the International Conference on Automated

Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-98), 1998.

53. Maus S., Moskal M., Schulte W. Vx86: x86 Assembler Simulated in C Powered by Automated Theorem Proving // Algebraic Methodology and Software Technology. 12th International Conference, AMAST 2008 Urbana, IL, USA, July 28-31, 2008. – pp. 284–298.

54. McCune W. Experiments with Discrimination-Tree indexing and Path Indexing for Term Retrieval // Journal of Automated Reasoning. 1992. V. 9(2). – pp. 147–167.

55. McCune W. Solution of the Robbins Problem // Journal of Automated Reasoning, V. 19(3), December 1997. – pp. 263–276.

56. Newell A., Shaw J.C. Programming the Logic Theory Machine // In Proceedings of the 1957 Western Joint Computer Conference, IRE, 1957. – pp.230–240.

57. Newell A., Shaw J.C., Simon H.A. Empirical Exploration With the Logic Theory Machine // Proceedings of the Western Joint Computer Conference, Vol. 15, 1957. – pp.218–239.

58. Plato J. A Constructive Theory of Ordered Affine Geometry // Indagationes Mathematicae. Volume 9, Issue 4, 21 December 1998. – pp. 549–562

59. Promsky A.V. Towards C-light Program Verification: Overcoming the Obstacles // Proc. International Workshop on Program Understanding, 19-23 June, Altai Mountains, Russia, 2009. – pp. 53–63.

60. Riazanov A. Implementing an Efficient Theorem Prover / PhD thesis, The University of Manchester, 2003. – 216 P.

61. Robinson J. A. A Machine–Oriented Logic Based on the Resolution Principle. / Robinson J. A. // Journal of the ACM. (12), 1965. pp. 23–41.

62. Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning. – MIT Press, Cambridge, 2001. – 2150p.

63. Schulz S. E — A Brainiac Theorem Prover // AI Communications - CASC. Volume 15 Issue 2,3, August 2002. pp. 111–126.

64. Schumann J. M. Automated Theorem Proving in Software Engineering / Springer, 2001. – 228p.

65. Sekar R., Ramakrishnan I.V., Voronkov A. Term Indexing. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 1853–1964. MIT Press, Cambridge. 2001.

66. Smullyan R.M. First-Order Logic / Dover Publications, New York. secong corrected edn, 1995. – 158p.

67. Smith D.E. A Source Book in Mathematics. – Dover Publications, 1984. – 701p.

68. Stikel M.E. Building theorem provers // Automated Deduction – CADE-22. 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. pp. 306–321.

69. Stickel M. The path-indexing method for indexing terms // Technical Note 473, Artificial Intelligence Center, SRI International, RAVENSWOOD AVE., MENLO PARK, CA 94025

70. Tammet T. Extending Classical Theorem Proving for the Semantic Web // In R. Volz, S. Decker, I. Cruz, editors, Proceedings of the First International Workshop on Practical and Scalable Semantic Systems, October 2003.

71. Turing A.M. On Computable Numbers, with an Application to the Entscheidungsproblem. - Proc.Lond. Math. Soc., Ser. 2, 1936, 42, – pp. 230–265, 1936, 43. – pp. 544–546.

72. Vassilyev, S.N. Machine Synthesis of Mathematical Theorems. The Journal of Logic programming, V.9, No.2–3, 1990. – pp. 235–266

73. Wang H. Toward Mechanical Mathematics // IBM J. Res. Devel., Vol. 4, No 1, 1960. – pp. 2–22.

74. Weidenbach C., Dimova D., Fietzke A., Kumar R., Suda M., Wischnewski P. SPASS Version 3.5 // Automated Deduction – CADE-22. 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings. – pp. 140–145.

75. Whitehead A.N., Russell B. Principia Mathematica. – Cambridge University Press, Cambridge, 2nd edition. January 2, 1927. – 1994p.

76. Wos L. Automated Reasoning: Introduction and Applications. / Wos L., Overbeek R., Lusk E., Boyle J. / McGraw–Hill, New York, 19. – 482p.

77. ACL2 Version 5.0 [Электронный ресурс]. URL: http://www.cs.utexas.edu/users/moore/acl2/ (дата обращения: 18.12.2012).

78. D Programming Language [Электронный ресурс]. URL: http://www.digitalmars.com/d/ (дата обращения: 18.12.2012).

79. E Prover [Электронный ресурс]. URL: http://www.eprover.org (дата обращения: 18.12.2012).

80. EQP [Электронный ресурс]. URL: http://www.cs.unm.edu/m̃ccune/eqp/ (дата обращения: 18.12.2012).

81. GLIB: Memory Slices [Электронный ресурс]. URL: http://developer.gnome.org/glib/2.34/glib-Memory-Slices.html (дата обращения: 18.12.2012).

82. iProver [Электронный ресурс]. URL: http://www.cs.man.ac.uk/ korovink/ /iprover/ (дата обращения: 18.12.2012).

83. Isabelle [Электронный ресурс]. URL: http://isabelle.in.tum.de/ (дата обращения: 18.12.2012).

84. The KeY Project [Электронный ресурс]. URL: http://www.key-project.org/ (дата обращения: 18.12.2012).

85. LeanCoP [Электронный ресурс]. URL: http://www.leancop.de (дата обращения: 18.12.2012).

86. Loprover [Электронный ресурс]. URL: http://www.ags.uni-sb.de/l̃eo/ (дата обращения: 18.12.2012).

87. MOLTAP — A Modal Logic Tableau Prover [Электронный ресурс]. URL: http://twan.home.fmf.nl/moltap/ (дата обращения: 18.12.2012).

88. Ontobox [Электронный ресурс]. URL: http://ontobox.org/ (дата обращения: 18.12.2012).

89. Otter [Электронный ресурс]. URL: http://www.cs.unm.edu/ mccune/otter/ (дата обращения: 18.12.2012).

90. The CADE ATP System Competition. The World Championship for Automated Theorem Proving [Электронный ресурс]. URL: http://www.cs.miami.edu/ tptp/CASC/ (дата обращения: 18.12.2012).

91. The Coq Proof Assistant [Электронный ресурс]. URL: http://coq.inria.fr/ (дата обращения: 18.12.2012).

92. The HOL Theorem Prover for Higher Order Logic [Электронный ресурс]. URL: http://www.cs.ox.ac.uk/tom.melham/res/hol.html (дата обращения: 18.12.2012).

93. Thousands of Problems for Theorem Provers [Электронный ресурс]. URL: http://tptp.org/ (дата обращения: 18.12.2012).

94. A.V.Gelder. TPTPparser utility. 2006 [Электронный ресурс]. URL: http://users.soe.ucsc.edu/~avg/TPTPparser/ (дата обращения: 18.12.2012).

95. Vampire [Электронный ресурс]. URL: http://www.vprover.org (дата обращения: 18.12.2012).

Приложение 1

Таблицы решенных задач из библиотеки TPTP

В данном приложении представлен список решенных задач (на 18.12.2012) из библиотеки TPTP. Для каждой задачи указаны количественные характеристики выражающие эффективность разработаной системы, согласно критериям: сложности решаемой задачи, времени, количества шагов.

Задачи геометрии (GEO)

Таблица П.1. – Задачи геометрии

Задача	**Рейтинг**	**Время**	**Шагов**
GEO205+3	0.33	0.31	3244
GEO195+2	0.29	13.04	177281
GEO202+3	0.29	0.02	96
GEO203+3	0.29	0.03	181
GEO177+2	0.25	0.52	6996
GEO193+2	0.25	13.4	177281
GEO194+2	0.25	12.65	177281
GEO197+2	0.25	0.23	15
GEO198+2	0.25	9.59	151322
GEO201+3	0.25	0.05	13
GEO205+2	0.25	0.09	10
GEO215+2	0.25	0.016	48
GEO222+1	0.25	0.06	13
GEO22+2	0.25	0.01	99
GEO172+3	0.21	0.37	2605
GEO174+3	0.21	0.29	2518
GEO180+1	0.21	3.83	51291
GEO181+1	0.21	3.5	51291
GEO184+3	0.21	0.006	17

Продолжение таблицы П.1.

Задача	**Рейтинг**	**Время**	**Шагов**
GEO187+3	0.21	0.6	1653
GEO191+2	0.21	4.42	55259
GEO191+3	0.21	0.01	38
GEO196+2	0.21	0.02	188
GEO203+2	0.21	4.77	51230
GEO205+1	0.21	0.02	7
GEO223+1	0.21	0.01	9
GEO230+1	0.21	0.006	8
GEO254+1	0.21	0.02	8
GEO255+3	0.21	0.05	389
GEO170+3	0.17	0.3	862
GEO177+3	0.17	0.92	6967
GEO179+2	0.17	0.47	7060
GEO180+2	0.17	0.44	6938
GEO182+2	0.17	0.45	6938
GEO186+3	0.17	0.005	17
GEO196+3	0.17	0.009	23
GEO197+3	0.17	0.006	8
GEO198+3	0.17	0.01	26
GEO199+3	0.17	0.01	13
GEO204+3	0.17	0.006	27
GEO210+2	0.17	0.009	25
GEO215+1	0.17	0.007	22
GEO221+1	0.17	0.01	9
GEO222+3	0.17	0.01	27

Продолжение таблицы П.1.

Задача	**Рейтинг**	**Время**	**Шагов**
GEO223+3	0.17	0.01	18
GEO259+1	0.17	0.01	8
GEO260+1	0.17	0.01	10
GEO172+1	0.12	0.2	2273
GEO173+3	0.12	0.01	28
GEO175+1	0.12	0.01	19
GEO175+2	0.12	7.01	90838
GEO176+3	0.12	0.31	2518
GEO179+3	0.12	0.004	26
GEO180+3	0.12	0.79	6480
GEO181+2	0.12	0.44	6938
GEO181+3	0.12	0.077	6480
GEO182+3	0.12	0.005	26
GEO183+3	0.12	1.79	10615
GEO185+3	0.12	0.005	6
GEO186+1	0.12	0.32	3669
GEO186+2	0.12	0.12	15
GEO188+3	0.12	0.02	97
GEO189+3	0.12	0.003	5
GEO190+3	0.12	0.07	281
GEO192+2	0.12	1.43	23456
GEO193+3	0.12	10.68	57668
GEO200+3	0.12	0.01	62
GEO201+1	0.12	0.07	1079
GEO201+2	0.12	0.07	1124

Продолжение таблицы П.1.

Задача	**Рейтинг**	**Время**	**Шагов**
GEO203+1	0.12	0.007	10
GEO204+1	0.12	0.03	285
GEO206+3	0.12	0.01	37
GEO217+1	0.12	0.5	5341
GEO218+3	0.12	0.019	50
GEO221+3	0.12	0.01	5
GEO224+3	0.12	0.68	6719
GEO225+2	0.12	0.26	3204
GEO225+3	0.12	0.05	134
GEO226+1	0.12	0.01	6
GEO226+2	0.12	0.01	57
GEO234+3	0.12	0.06	393
GEO242+1	0.12	0.03	11
GEO266+3	0.12	0.33	12
GEO170+2	0.08	0.05	728
GEO171+1	0.08	0.64	7955
GEO172+2	0.08	6.67	90480
GEO173+1	0.08	0.006	9
GEO173+2	0.08	0.41	6470
GEO174+1	0.08	0.02	336
GEO175+1	0.08	0.01	38
GEO183+1	0.08	1.9	26359
GEO184+1	0.08	0.03	9
GEO184+2	0.08	0.4	6470
GEO192+3	0.08	1.8	12129

Продолжение таблицы П.1.

Задача	**Рейтинг**	**Время**	**Шагов**
GEO200+1	0.08	0.01	42
GEO206+1	0.08	0.01	33
GEO207+1	0.08	0.001	2
GEO208+1	0.08	0.01	9
GEO209+1	0.08	0.08	45
GEO209+2	0.08	0.01	76
GEO209+3	0.08	0.06	242
GEO210+1	0.08	0.01	19
GEO210+3	0.08	0.007	17
GEO212+2	0.08	0.01	8
GEO213+1	0.08	0.01	8
GEO213+2	0.08	0.01	8
GEO216+1	0.08	0.01	4
GEO217+2	0.08	1.44	19097
GEO217+3	0.08	0.01	26
GEO218+1	0.08	0.01	6
GEO218+2	0.08	0.01	6
GEO219+1	0.08	0.01	10
GEO219+3	0.08	0.01	5
GEO220+3	0.08	0.003	14
GEO225+1	0.08	0.01	6
GEO226+3	0.08	0.62	2622
GEO227+3	0.08	0.11	725
GEO231+3	0.08	0.04	298
GEO232+3	0.08	0.05	296

Продолжение таблицы П.1.

Задача	**Рейтинг**	**Время**	**Шагов**
GEO233+3	0.08	0.07	495
GEO234+3	0.08	0.03	202
GEO235+3	0.08	0.009	44
GEO236+3	0.08	0.003	44
GEO238+3	0.08	0.03	212
GEO239+3	0.08	0.03	42
GEO240+3	0.08	0.05	373
GEO241+1	0.08	0.03	12
GEO242+3	0.08	0.07	586
GEO243+3	0.08	0.03	42
GEO244+3	0.08	0.03	299
GEO247+3	0.08	0.01	42
GEO248+3	0.08	0.07	496
GEO249+3	0.08	0.03	199
GEO250+3	0.08	0.01	42
GEO251+3	0.08	0.01	198
GEO252+3	0.08	0.01	42
GEO253+3	0.08	0.03	198
GEO255+1	0.08	0.01	13
GEO257+3	0.08	0.03	22
GEO258+3	0.08	0.24	1927
GEO261+1	0.08	0.05	209
GEO261+3	0.08	0.1	770
GEO262+3	0.08	0.1	12
GEO263+1	0.08	0.05	249

Продолжение таблицы П.1.

Задача	**Рейтинг**	**Время**	**Шагов**
GEO263+3	0.08	0.19	11
GEO264+3	0.08	0.11	12
GEO265+3	0.08	0.1	12
GEO268+3	0.08	0.17	11
GEO269+3	0.08	0.56	12
GEO170+1	0.04	0.01c	8
GEO171+2	0.04	0.002c	5
GEO171+3	0.04	2.04	14336
GEO174+2	0.04	6.21	88441
GEO176+1	0.04	0.34	4384
GEO176+2	0.04	0.02	236
GEO178+1	0.04	0.005	20
GEO178+2	0.04	0.001	20
GEO178+3	0.04	0.11	977
GEO185+1	0.04	0.32	4364
GEO185+2	0.04	0.007	42
GEO204+2	0.04	0.01	6
GEO206+2	0.04	0.005	35
GEO207+2	0.04	0.001	2
GEO207+3	0.04	0.005	19
GEO208+2	0.04	0.01	9
GEO208+3	0.04	0.02	81
GEO211+1	0.04	0.001	6
GEO211+2	0.04	0.01	4
GEO212+1	0.04	0.005	27

Продолжение таблицы П.1.

Задача	**Рейтинг**	**Время**	**Шагов**
GEO214+2	0.04	0.01	16
GEO216+2	0.04	0.01	4
GEO216+3	0.04	0.04	94
GEO219+2	0.04	0.01	12
GEO220+1	0.04	0.01	29
GEO220+2	0.04	0.01	18
GEO228+1	0.04	0.27	1087
GEO229+1	0.04	0.001	1
GEO233+1	0.04	0.009	13
GEO236+1	0.04	0.01	11
GEO237+3	0.04	0.05	296
GEO238+1	0.04	0.03	97
GEO239+1	0.04	0.03	11
GEO240+1	0.04	0.03	11
GEO241+3	0.04	0.008	16
GEO243+1	0.04	0.03	11
GEO244+1	0.04	0.03	11
GEO245+1	0.04	0.03	11
GEO245+3	0.04	0.01	16
GEO246+1	0.04	0.03	11
GEO246+3	0.04	0.01	14
GEO247+1	0.04	0.03	11
GEO248+1	0.04	0.03	11
GEO249+1	0.04	0.03	11
GEO250+1	0.04	0.03	12

Продолжение таблицы П.1.

Задача	**Рейтинг**	**Время**	**Шагов**
GEO251+1	0.04	0.03	11
GEO252+1	0.04	0.03	11
GEO256+1	0.04	0.03c	11
GEO256+3	0.04	0.006	16
GEO257+1	0.04	0.03c	11
GEO258+1	0.04	0.005c	13
GEO262+1	0.04	0.05c	252
GEO264+1	0.04	0.05c	11
GEO267+3	0.04	0.38c	12
GEO214+1	0	0.01	8
GEO227+1	0	0.001	36
GEO232+1	0	0.01	11
GEO235+1	0	0.01	11
GEO237+1	0	0.01	18
GEO253+1	0	0.03	17

Задачи Common sense reasoning (CSR)

Таблица П.2. – Задачи common sense reasoning

Задача	**Рейтинг**	**Время**	**Шагов**
CSR038+2	0.33	0.6	188
CSR066+2	0.33	0.6	188
CSR027+2	0.29	37.6	10060
CSR033+2	0.29	86.98	17119
CSR070+2	0.29	11.7	3228

Продолжение таблицы П.2.

Задача	**Рейтинг**	**Время**	**Шагов**
CSR073+2	0.29	19.4	4607
CSR057+2	0.25	68.3	10483
CSR062+2	0.25	127.6	15141
CSR029+1	0.22	0.03	130
CSR042+1	0.22	0.01	46
CSR029+2	0.21	5.9	1969
CSR030+2	0.21	7.8	2466
CSR032+2	0.21	4.8	1546
CSR056+2	0.21	7.5	2310
CSR065+2	0.21	12.8	3466
CSR069+3	0.21	0.006	16
CSR071+2	0.21	5.12	1731
CSR074+2	0.21	11.1	3095
CSR025+3	0.17	0.0002	1
CSR026+2	0.17	13.4	3516
CSR041+2	0.17	0.12	26
CSR042+2	0.17	2.7	848
CSR044+1	0.17	0.02	118
CSR037+2	0.12	0.4	145
CSR031+2	0.12	0.8	197
CSR032+1	0.12	0.006	37
CSR035+1	0.12	0.006	25
CSR051+2	0.12	1.2	248
CSR053+2	0.12	0.09	10
CSR054+2	0.12	1.4	304

Продолжение таблицы П.2.

Задача	**Рейтинг**	**Время**	**Шагов**
CSR058+2	0.12	1.3	272
CSR035+2	0.12	1.4	416
CSR038+1	0.12	0.006	44
CSR040+1	0.12	0.36	711
CSR050+1	0.12	0.05	204
CSR054+1	0.12	0.004	34
CSR055+1	0.12	0.004	63
CSR058+1	0.12	0.004	25
CSR066+1	0.12	0.004	53
CSR059+2	0.12	4.4	1545
CSR025+1	0.11	0.0002	1
CSR057+1	0.11	0.12	455
CSR059+1	0.11	0.004	64
CSR048+1	0.11	0.02	124
CSR067+1	0.11	0.004	46
CSR071+1	0.11	0.04	109
CSR074+1	0.11	0.04	130
CSR025+2	0.08	0.0002	1
CSR026+1	0.08	0.001	64
CSR039+1	0.08	4.8	5993
CSR055+2	0.08	0.8	197
CSR069+2	0.08	0.09	20
CSR043+2	0.08	0.001	1
CSR046+2	0.08	0.001	1
CSR052+1	0.08	0.39	1033

Продолжение таблицы П.2.

Задача	**Рейтинг**	**Время**	**Шагов**
CSR041+1	0.08	0.02	62
CSR068+1	0.08	0.004	30
CSR070+1	0.08	0.02	211

Задачи менеджмента (MGT)

Таблица П.3. – Задачи менеджмента

Задача	**Рейтинг**	**Время**	**Шагов**
MGT025+1	0.26	0.25	1552
MGT004+1	0.17	0.01	11
MGT001+1	0.12	0.01	20
MGT003+1	0.12	0.01	11
MGT006+1	0.12	0.002	16
MGT007+1	0.12	0.002	18
MGT008+1	0.12	0.002	6
MGT009+1	0.12	0.002	8
MGT015+1	0.12	0.1	7
MGT016+1	0.12	0.01	7
MGT017+1	0.12	0.1	7
MGT018+1	0.12	0.01	7
MGT036+1	0.12	0.05	11
MGT036+2	0.12	0.04	10
MGT002+1	0.08	0.01	12
MGT028+1	0.08	0.001	30
MGT021+1	0.04	0.01	11

Продолжение таблицы П.3.

Задача	**Рейтинг**	**Время**	**Шагов**
MGT022+1	0.04	0.01	9
MGT022+2	0.04	0.01	9
MGT024+1	0.04	0.01	108
MGT032+2	0.04	0.002	10
MGT036+3	0.04	0.001	16
MGT041+2	0	0.001	23
MGT045+1	0	0.04	278
MGT052+1	0	0.04	114

Задачи синтаксиса (SYN)

Таблица П.4. – Задачи синтаксиса

Задача	**Рейтинг**	**Время**	**Шагов**
SYN353+1	0.54	0.002	33
SYN548+1	0.54	0.01	15
SYN549+1	0.29	0.07	902
SYN938+1	0.29	0.08	13
SYN321-1	0.20	0.001	6
SYN550+1	0.17	0.004	51
SYN386+1	0.17	0.02	15
SYN380+1	0.12	0.001	6
SYN947+1	0.12	0.0002	3
SYN404+1	0.11	0.0005	2
SYN928+1	0.11	0.0002	2
SYN918+1	0.08	0.0005	5

Продолжение таблицы П.4.

Задача	**Рейтинг**	**Время**	**Шагов**
SYN732+1	0.08	0.0005	5
SYN939+1	0.08	0.0002	13
SYN943+1	0.08	0.0002	23
SYN940+1	0.04	0.0002	15
SYN065+1	0.04	0.0008	11
SYN066+1	0.04	0.0002	22
SYN073+1	0.04	0.0002	6
SYN733+1	0.04	0.0005	5
SYN948+1	0.04	0.0002	2
SYN050+1	0	0.001	6
SYN052+1	0	0.0002	17
SYN054+1	0	0.008	75
SYN055+1	0	0.002	14
SYN056+1	0	0.002	33
SYN057+1	0	0.0002	13
SYN058+1	0	0.0002	15
SYN059+1	0	0.0002	20
SYN060+1	0	0.0002	7
SYN062+1	0	0.0002	7
SYN068+1	0	0.0002	16
SYN069+1	0	0.0002	28
SYN070+1	0	0.0002	21
SYN079+1	0	0.0002	2
SYN082+1	0	0.0002	7
SYN401+1	0	0.0005	2

Продолжение таблицы П.4.

Задача	**Рейтинг**	**Время**	**Шагов**
SYN402+1	0	0.0005	2
SYN403+1	0	0.0005	3
SYN406+1	0	0.0005	6
SYN407+1	0	0.0005	9
SYN408+1	0	0.0005	2
SYN410+1	0	0.0005	2
SYN411+1	0	0.0005	1
SYN412+1	0	0.0005	4
SYN413+1	0	0.004	35
SYN919+1	0	0.0002	7
SYN920+1	0	0.0002	11
SYN923+1	0	0.0002	3
SYN926+1	0	0.0002	4
SYN927+1	0	0.0002	2
SYN929+1	0	0.0002	3
SYN942+1	0	0.0002	9
SYN944+1	0	0.0002	5
SYN949+1	0	0.0002	5
SYN950+1	0	0.0002	7
SYN952+1	0	0.0002	2
SYN953+1	0	0.0002	4
SYN954+1	0	0.0002	7
SYN955+1	0	0.0002	3
SYN956+1	0	0.0002	4
SYN957+1	0	0.0002	4

Продолжение таблицы П.4.

Задача	**Рейтинг**	**Время**	**Шагов**
SYN958+1	0	0.0002	6
SYN959+1	0	0.0002	3
SYN961+1	0	0.0002	5
SYN962+1	0	0.0002	5
SYN964+1	0	0.0002	3
SYN969+1	0	0.0002	3
SYN970+1	0	0.0002	3
SYN972+1	0	0.0002	4
SYN973+1	0	0.0002	1
SYN974+1	0	0.0002	2
SYN975+1	0	0.0002	3
SYN976+1	0	0.0002	2
SYN977+1	0	0.0002	2
SYN978+1	0	0.0002	4
SYN979+1	0	0.0002	9
SYN981+1	0	0.0002	6
SYN986+1.000	0	0.0002	4

Задачи семантического веба (SWB)

Таблица П.5. – Задачи семантического веба

Задача	**Рейтинг**	**Время**	**Шагов**
SWB012+3	0.54	65.64	102428
SWB020+2	0.5	0.6	618
SWB029+3	0.38	55.27	98471

Продолжение таблицы П.5.

Задача	**Рейтинг**	**Время**	**Шагов**
SWB014+3	0.33	60.06	106891
SWB003+1	0.3	0.01	1
SWB009+3	0.25	62.41	102428
SWB027+2	0.22	0.08	15
SWB022+2	0.21	0.02	10
SWB012+2	0.21	1.5	10
SWB025+2	0.17	0.02	11
SWB002+3	0.08	0.4	1042
SWB014+2	0.08	0.06	1088
SWB029+2	0.08	0.02	13
SWB001+3	0.04	0.03	43
SWB003+3	0.04	0.01	1
SWB003+4	0.04	0.002	1
SWB024+2	0.04	0.06	1195
SWB001+2	0	0.0002	3
SWB001+4	0	0.006	18
SWB002+2	0	0.004	8
SWB002+4	0	0.01	47
SWB003+2	0	0.01	1

Другие задачи

Таблица П.6. – Другие задачи

Задача	**Рейтинг**	**Время**	**Шагов**
LCL652+1.015	0.92	32.1	185253

Продолжение таблицы П.6.

Задача	**Рейтинг**	**Время**	**Шагов**
LCL656+1.020	0.92	1.24	845
LCL656+1.015	0.88	0.77	845
LCL660+1.010	0.79	1.98	17745
LCL660+1.015	0.79	35.4	320457
KRS251+1	0.79	5.09	26406
LCL660+1.005	0.75	1.31	14880
LCL640+1.010	0.67	8.00	65876
LCL640+1.005	0.62	0.06	821
COM008+1	0.62	1.9	33551
LCL656+1.010	0.58	1.5	43
LCL648+1.005	0.54	4.02	23898
LCL636+1.005	0.5	0.5	20
LCL666+1.005	0.5	0.35	2592
KRS235+1	0.46	5.4	24406
LCL642+1.001	0.46	231.6	2128225
LCL674+1.020	0.42	5.1	577
COM003+1	0.33	0.01	10
LCL656+1.005	0.25	0.2	23
TOP005-2	0.25	0.49	4561
SWV323-2	0.25	0.03	212
COM003+2	0.21	0.8	20
COM003+3	0.17	0.01	10
ALG211+1	0.12	0.027	644
PUZ035-3	0.12	0.008	58
SWV329-2	0.12	0.005	13

Продолжение таблицы П.6.

Задача	**Рейтинг**	**Время**	**Шагов**
MSC002-2	0.12	52.2	776071
PUZ031+1	0.08	0.005	19
PUZ031+3	0.08	0.4	19
PUZ047+1	0.08	0.01	12
SET044+1	0.08	0.002	7
SET062+4	0.07	0.002	20
SET045+1	0.04	0.002	14
PUZ031+2	0.04	0.02	19
SET062+3	0.04	0.002	9
SET913+1	0.04	0.002	6
SEU158+3	0.04	0.01	10
SEU163+1	0.04	0.01	5
SEU163+4	0.04	0.02	21
SEU167+3	0.04	0.03	20
TOP021+1	0.04	0.01	6
SET043+1	0	0.0002	4
SET046+1	0	0.002	22
SET574+3	0	0.02	486
SET575+3	0	0.02	10
SET576+3	0	0.16	2780
SET631+3	0	0.068	1319
SET915+1	0	0.002	6
SEU261+1	0	0.004	44
SEU275+1	0	0.005	34
MSC009+1	0	0.08	1

Продолжение таблицы П.6.

Задача	**Рейтинг**	**Время**	**Шагов**
SWV010+1	0	0.008	1
MSC004-1	0	0.02	100
ANA042-2	0	0.0006	7
TOP004-2	0	0.0006	4
TOP004-1	0	0.003	7
TOP001-2	0	0.25	2901

Задачи FOF без равенства с чемпионата мира 2012 среди систем АДТ, для которых был найден логический вывод

В таблице П.7. представлены решенные задачи.

Таблица П.7. – Задачи CASC-J6

Задача	**Результат (секунды)**
CSR069+3	0.8
GEO230+1	0.006
SWB012+2	1.5
COM003+2	0.8
SWB022+2	0.02
GEO203+2	4.77
SYN917+1	0.001
CSR030+2	7.8
SWB009+2	62.41
GEO293+2	13.41
GEO198+2	9.59
CSR057+2	68.3

Продолжение таблицы П.7.

Задача	**Результат (секунды)**
CSR062+2	127.6
GEO201+3	0.05
SYN938+1	0.08
CSR027+2	37.6
CSR033+2	86.98
SWB014+3	60.2
COM003+1	0.01
LCL674+1.020	5.1
LCL666+1.005	0.35
SWB020+2	0.6
LCL648+1.005	4.02
SWB012+3	65.64
LCL640+1.010	8.00
LCL652+1.015	32.1

Приложение 2

Пример формализации задачи в языке ПО–формул

Задача COM003+1 из библиотеки TPTP в языке ПО–формул имеет следующий вид:

```
{
  e[X1][algorithm(X1)] {
    a[W_Iy4][program(W_Iy4)] {
      e[Y_62o2,Z_62o2][program(Y_62o2)] {
        a[][decides(W_Iy4,Y_62o2,Z_62o2)] {
          e[][False] {}}};
      e[][] {
        a[Y_uQg2,Z_uQg2][] {
          e[][] {
            a[][program(Y_uQg2),halts2(Y_uQg2,Z_uQg2)] {
              e[][halts3(W_Iy4,Y_uQg2,Z_uQg2),outputs(W_Iy4,
good),] {}};
            a[][program(Y_uQg2)] {
              e[][halts2(Y_uQg2,Z_uQg2)] {};
              e[][halts3(W_Iy4,Y_uQg2,Z_uQg2),outputs(W_Iy4,
bad),] {}}}}}};
    a[][] {
      e[][] {
        a[X][] {
          e[][] {
            a[][algorithm(X)] {
              e[][False] {}}};
          e[Y,Z][program(Y)] {
            a[][decides(X,Y,Z)] {
              e[][False] {}}}}};
      e[W][program(W)] {
```

```
    a[Y_q4E1][program(Y_q4E1)] {
      e[][] {
        a[Z_q4E1][] {
          e[][decides(W,Y_q4E1,Z_q4E1)] {}}}}}};
a[][] {
  e[][] {
    a[W_MFP2][] {
      e[][] {
        a[][program(W_MFP2)] {
          e[][False] {}}};
      e[Y_pmk][] {
        a[][] {
          e[][program(Y_pmk),halts2(Y_pmk,Y_pmk)] {
            a[][] {
              e[][] {
                a[][halts3(W_MFP2,Y_pmk,Y_pmk)] {
                  e[][False] {}}};
              e[][] {
                a[][outputs(W_MFP2,good)] {
                  e[][False] {}}}}};
            e[][program(Y_pmk)] {
              a[][halts2(Y_pmk,Y_pmk)] {
                e[][False] {}};
              a[][] {
                e[][] {
                  a[][halts3(W_MFP2,Y_pmk,Y_pmk)] {
                    e[][False] {}}};
                e[][] {
                  a[][outputs(W_MFP2,bad)] {
```

```
                    e[][False] {}}}}}}}};
  e[V][program(V)] {
    a[Y_MFP2][] {
      e[][] {
        a[][program(Y_MFP2),halts2(Y_MFP2,Y_MFP2)] {
          e[][halts2(V,Y_MFP2),outputs(V,good),] {}};
        a[][program(Y_MFP2)] {
          e[][halts2(Y_MFP2,Y_MFP2)] {};
          e[][halts2(V,Y_MFP2),outputs(V,bad),] {}}}}};
a[][] {
  e[][] {
    a[V_t382][] {
      e[][] {
        a[][program(V_t382)] {
          e[][False] {}}};
      e[Y_5hG2][] {
        a[][] {
          e[][program(Y_5hG2),halts2(Y_5hG2,Y_5hG2)] {
            a[][] {
              e[][] {
                a[][halts2(V_t382,Y_5hG2)] {
                  e[][False] {}}};
              e[][] {
                a[][outputs(V_t382,good)] {
                  e[][False] {}}}};
            e[][program(Y_5hG2)] {
              a[][halts2(Y_5hG2,Y_5hG2)] {
                e[][False] {}};
              a[][] {
```

```
                    e[][] {
                      a[][halts2(V_t382,Y_5hG2)] {
                        e[][False] {}}};
                    e[][] {
                      a[][outputs(V_t382,bad)] {
                        e[][False] {}}}}}}}}};
      e[U][program(U)] {
        a[Y_HbT][] {
          e[][] {
            a[][program(Y_HbT),halts2(Y_HbT,Y_HbT),
halts2(U,Y_HbT)] {
              e[][False] {}};
            a[][program(Y_HbT)] {
              e[][halts2(Y_HbT,Y_HbT)] {};
              e[][halts2(U,Y_HbT),outputs(U,bad),] {}}}}}};
    a[Y1][program(Y1)] {
      e[][] {
        a[Z1][] {
          e[][decides(X1,Y1,Z1)] {}}}}}
}
```

Приложение 3

Пример сокращённого протокола вывода

Пример сокращённого протокола вывода для задачи MSC004-1 из библиотеки TPTP:

```
e[in(john,boy)0,]
    a[Big_part, Number_of_mid_parts, Mid_part, Small_part,
 Number_of_small_parts, ]
    a[has_parts(Big_part,Number_of_mid_parts,Mid_part),
has_parts(object_in(Big_part,Mid_part,Small_part,
Number_of_mid_parts,Number_of_small_parts),
Number_of_small_parts,Small_part),]
        e[]
        e[has_parts(Big_part,times(Number_of_mid_parts,
Number_of_small_parts),Small_part),]


    a[X, ]
    a[in(X,boy),]
        e[]
        e[in(X,human),]


    a[X, ]
    a[in(X,hand),]
        e[]
        e[has_parts(X,n5,fingers),]


    a[X, ]
    a[in(X,human),]
```

```
        e[]
        e[has_parts(X,n2,arm),]


    a[X, ]
    a[in(X,arm),]
        e[]
        e[has_parts(X,n1,hand),]


    a[]
    a[has_parts(john,times(times(n2,n1),n5),fingers),]
        e[]
        e[False,]


    a[Big_part, Number_of_mid_parts, Mid_part, Small_part,
 Number_of_small_parts, ]
    a[has_parts(Big_part,Number_of_mid_parts,Mid_part),]
        e[]
        e[has_parts(Big_part,times(Number_of_mid_parts,
Number_of_small_parts),Small_part),]

        e[]
        e[in(object_in(Big_part,Mid_part,Small_part,
Number_of_mid_parts,Number_of_small_parts),Mid_part),]
```

```
{X -> john, }
{X -> john, }
{Big_part -> john, Number_of_mid_parts -> n2, Mid_part -> arm,
 Small_part -> h1, Number_of_small_parts -> h2, }
{Big_part -> john, Number_of_mid_parts -> times(n2,h2),
 Mid_part -> h1, Small_part -> h3, Number_of_small_parts ->
 h4, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]


{h2 -> n1, h4 -> n5, h3 -> fingers, }
{X -> object_in(john,boy,fingers,times(n2,n1),n5),
 h1 -> boy, }
{X -> object_in(john,boy,fingers,times(n2,n1),n5), }
{Big_part -> object_in(john,boy,fingers,times(n2,n1),n5),
 Number_of_mid_parts -> n2, Mid_part -> arm, Small_part ->
 h5, Number_of_small_parts -> h6, }
{Big_part -> object_in(john,boy,fingers,times(n2,n1),n5),
 Number_of_mid_parts -> times(n2,h6), Mid_part -> h5,
 Small_part -> h7, Number_of_small_parts -> h8, }
{Big_part -> object_in(john,boy,fingers,times(n2,n1),n5),
 Number_of_mid_parts -> times(times(n2,h6),h8), Mid_part ->
 h7, Small_part -> h9, Number_of_small_parts -> h10, }
{Big_part -> object_in(john,boy,fingers,times(n2,n1),n5),
 Number_of_mid_parts -> times(times(times(n2,h6),h8),h10),
 Mid_part -> h9, Small_part -> h11,
```

```
 Number_of_small_parts -> h12, }
{Big_part -> object_in(john,boy,fingers,times(n2,n1),n5),
 Number_of_mid_parts -> times(times(times(times(n2,h6),h8),
h10),h12), Mid_part -> h11, Small_part -> h13,
 Number_of_small_parts -> h14, }
{Big_part -> object_in(john,boy,fingers,times(n2,n1),n5),
 Number_of_mid_parts -> times(times(times(times(times(n2,h6),
h8),h10),h12),h14), Mid_part -> h13, Small_part -> h15,
 Number_of_small_parts -> h16, }
{Big_part -> object_in(john,boy,fingers,times(n2,n1),n5),
 Number_of_mid_parts -> times(times(times(times(times
(times(n2,h6),h8),h10),h12),h14),h16), Mid_part -> h15,
 Small_part -> h17, Number_of_small_parts -> h18, }
{X -> object_in(john,hand,fingers,times(n2,n1),n5),
 h1 -> hand, }
{Big_part -> object_in(john,hand,fingers,times(n2,n1),n5),
 Number_of_mid_parts -> n5, Mid_part -> fingers,
 Small_part -> h19, Number_of_small_parts -> h20, }
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
 Mid_part -> hand, Small_part -> fingers,
 Number_of_small_parts -> n5, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]



{}
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
```

```
 Mid_part -> hand, Small_part -> fingers,
 Number_of_small_parts -> n5, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]



{}
{X -> object_in(john,arm,hand,n2,n1), }
{Big_part -> object_in(john,arm,hand,n2,n1),
 Number_of_mid_parts -> n1, Mid_part -> hand,
 Small_part -> h21, Number_of_small_parts -> h22, }
{Big_part -> john, Number_of_mid_parts -> n2,
 Mid_part -> arm, Small_part -> hand,
 Number_of_small_parts -> n1, }
{Big_part -> object_in(john,arm,hand,n2,n1),
 Number_of_mid_parts -> times(n1,h22), Mid_part -> h21,
 Small_part -> h23, Number_of_small_parts -> h24, }
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
 Mid_part -> hand, Small_part -> h25,
 Number_of_small_parts -> h26, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]



{h26 -> n5, h25 -> fingers, }
```

```
{X -> object_in(john,hand,fingers,times(n2,n1),n5), }
{Big_part -> object_in(john,arm,hand,n2,n1),
 Number_of_mid_parts -> times(times(n1,h22),h24), Mid_part ->
 h23, Small_part -> h27, Number_of_small_parts -> h28, }
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
 Mid_part -> hand, Small_part -> fingers,
 Number_of_small_parts -> n5, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]


{}
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
 Mid_part -> hand, Small_part -> fingers,
 Number_of_small_parts -> n5, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]


{}
{X -> object_in(object_in(john,arm,hand,n2,n1),boy,h23,
times(n1,h22),h24), h21 -> boy, }
{X -> object_in(object_in(john,arm,hand,n2,n1),boy,h23,
times(n1,h22),h24), }
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
```

```
 Mid_part -> hand, Small_part -> h29,
 Number_of_small_parts -> h30, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]


{h30 -> n5, h29 -> fingers, }
{Big_part -> object_in(john,arm,hand,n2,n1),
 Number_of_mid_parts -> times(n1,h22), Mid_part -> boy,
 Small_part -> arm, Number_of_small_parts -> n2, h24 -> n2,
 h23 -> arm, }
{X -> object_in(john,hand,fingers,times(n2,n1),n5), }
{Big_part -> object_in(object_in(john,arm,hand,n2,n1),boy,
arm,times(n1,h22),n2), Number_of_mid_parts -> n2,
 Mid_part -> arm, Small_part -> h31,
 Number_of_small_parts -> h32, }
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
 Mid_part -> hand, Small_part -> fingers,
 Number_of_small_parts -> n5, }
{Big_part -> object_in(object_in(john,arm,hand,n2,n1),boy,
arm,times(n1,h22),n2), Number_of_mid_parts -> n2,
 Mid_part -> arm, Small_part -> h33,
 Number_of_small_parts -> h34, }
{Big_part -> object_in(john,arm,hand,n2,n1),
 Number_of_mid_parts -> times(times(n1,h22),n2),
 Mid_part -> arm, Small_part -> h35,
 Number_of_small_parts -> h36, }
```

```
{Big_part -> object_in(object_in(john,arm,hand,n2,n1),boy,
arm,times(n1,h22),n2), Number_of_mid_parts -> times(n2,h34),
 Mid_part -> h33, Small_part -> h37, Number_of_small_parts ->
 h38, }
{X -> object_in(john,hand,fingers,times(n2,n1),n5), }
{Big_part -> object_in(object_in(john,arm,hand,n2,n1),boy,h23,
times(n1,h22),h24), Number_of_mid_parts -> n2, Mid_part ->
 arm, Small_part -> h39, Number_of_small_parts -> h40, }
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
 Mid_part -> hand, Small_part -> fingers,
 Number_of_small_parts -> n5, }
{Big_part -> object_in(john,hand,fingers,times(n2,n1),n5),
 Number_of_mid_parts -> n5, Mid_part -> fingers,
 Small_part -> h41, Number_of_small_parts -> h42, }
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
 Mid_part -> hand, Small_part -> fingers,
 Number_of_small_parts -> n5, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]



{}
{Big_part -> john, Number_of_mid_parts -> n2, Mid_part -> arm,
 Small_part -> hand, Number_of_small_parts -> n1, }
{X -> object_in(object_in(john,arm,hand,n2,n1),hand,boy,n1,
h22), }
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
```

```
 Mid_part -> hand, Small_part -> h43,
 Number_of_small_parts -> h44, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]


{h44 -> n5, h43 -> fingers, }
{X -> object_in(john,hand,fingers,times(n2,n1),n5), }
{Big_part -> object_in(object_in(john,arm,hand,n2,n1),hand,
boy,n1
,h22), Number_of_mid_parts -> n5, Mid_part -> fingers,
 Small_part -> h45, Number_of_small_parts -> h46, }
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
 Mid_part -> hand, Small_part -> fingers,
 Number_of_small_parts -> n5, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
    e[]
    e[False,]


{}
{Big_part -> john, Number_of_mid_parts -> times(n2,n1),
 Mid_part -> hand, Small_part -> fingers,
 Number_of_small_parts -> n5, }
a[]
a[has_parts(john,times(times(n2,n1),n5),fingers),]
```

```
    e[]
    e[False,]


{}
==========ok==========
sc: 70
.........................
Time: 0.006316s.
================================
```

Приложение 4

Справки о внедрении результатов работы