УЧРЕЖДЕНИЕ РОССИЙСКОЙ АКАДЕМИИ НАУК
ИНСТИТУТ ДИНАМИКИ СИСТЕМ И ТЕОРИИ УПРАВЛЕНИЯ
СИБИРСКОГО ОТДЕЛЕНИЯ РАН

На правах рукописи
УДК: 004.4'24:004.896

Ларионов Александр Александрович

Программная система автоматического доказательства теорем в исчислении позитивно-образованных формул

05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель
к.т.н. Е.А. Черкашин

Иркутск — 2012

Содержание

Глава 1. Введение

1.1. Об автоматизации рассуждений

1.1.1. Исторический взгляд

Пионерские идеи об автоматизации (механизации) рассуждений скорее всего высказали Раймунд Луллий(1235-1315) и позднее Готфрид Лейбниц (1646-1716). 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 бы механизированно 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 некоего 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 алгеб-

ры; Готтлоб Фреге [**Frege**], [**Sourcebook**], 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 [**GilbertAkkerman**]; Курт Гёдель, Алан Тьюринг, Алонзо Черч, получили результаты, касающиеся пределов возможностей формальных систем, в частности полнота [**Godel1929**] 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 совокупнсоти 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 обсуловлено выживанием в условиях второй мировой войны, можно отметить работы Джона фон Неймана, Норберта Винера, Алана Тьюринга, Конрада Цузе.

Развитие апарата математической логики и вычислительных машан естественно привело к созданию первых работающих на практике систем автоматического доказателсьтва теорем: 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) [**LogicComp**]; «Логик-теоретик», разработанный А. Невелом, Г. Саймоном, Дж. К. Шоу [**Newell1**], [**Newell2**] в 1956 году для доказательства некоторых задач из Principia Mathematica [**PrinMat**], причем данная система была направлена на моделирование человеческих рассуждений; В 1958 г. Ван Хао создаёт систему, доказавшую 350 задач из Principia Mathematica [**WangHao**].

Началом сильного развития области АДТ явился метод резолюций [**Robinson_1965**] предложенный Дж. Робинсоном в 1965 году (работы ве-

лись совместно с Д. Карсоном и Л. Уосом). Причиной его успеха явилась достаточно хорошая пригодность формализма для реализации на компьютере, в частности однородность представления данных и единственность правила вывода. Стоит отметить что данный метод и по сей день занимает доминирующее пололжение среди теоретического базиса для систем АДТ.

Отметим что уже в то время указывалось на важность применения эвристик []. И уже тогда зародилась некоторая конкуренция между чисто машинными подходами и эвристическими. В 1960-ые Л.М. Нортон разработал эвристический прувер для теории групп [ЛиК5]. [Надо ещё примеров].

В 1994 годы системой EQP была доказана открытая математическая проблема, что сильно повысило планку возможностей пруверов.

Добавить про современное состояние дел.

1.1.2. Область применения систем АДТ

Первые системы АДТ предназначались скорее для подтверждения возможности автоматизации рассуждений а также для удовлетворения спортивного интереса авторов. Чуть позднее с появлением логического программирования, интерес перешел в сферу некоторых задачек ИИ (однако это уже не совсем АДТ).

На сегодняшней день использование АДТ (с обоснованием его эффективности) замечено в следующих областях: верификация программ [], синтез программ [**Butakov1**], верификация оборудования [], обработки естественных языков [**ATP_NLP**], решения задачек типа оригами, сокобана [**Origami**], в исследовании protocol languages [], исследование безопасности информационных потоков [**ATP_Flow**], view deletion в базах данных [**ATP_DB**], семантическом вебе [], составлении расписаний [], задачах управления [], и даже компьютерном зрении [**ATP_Vision**] и др.

Наибольшую популярность имеют: верификация программных и аппаратных систем; синтез программного обеспечения; решение некоторых проблем математики (библиотека ТПТП); логическое программирование; дедуктивные базы данных.

1.1.3. Современные пруверы, проблематика и актуальность

Мы классифицируем системы автоматизации логического вывода следующим образом:

1. Классические. Предназначены для автоматического доказательств теорем, выраженных 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, Vampire (надо добавить, что вампир может компилировать задачу, но он не использет никакой содержательной инфы о задаче, просто структуры данных «статические» и проиндексированные получаются), E, SPASS, EQP, Prover9 и др. В частности с помощью EQP была доказана открытая математическая проблема [], а Vampire уже много лет является победителем турнира среди систем АДТ []. Кроме того, общей чертой данных систем, является использование лишь синтаксической информации о решаемой задаче.

2. Системы, предназначенные для заранее определенного класса задач и неклассических логик. Например для различных алгебраических систем [], геометрические пруверы [] и др. Характерны тем что либо в принципе предназначены только для указанного класса, либо показывают хорошую производительность на задачах такого 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 или скажем для конструктивных логик или модальных.

3. Настраиваемые (полиморфные) системы. От части к таким системам можно отнести и системы из предыдущих классов, однако под настраиванием мы понимаем в большей степени настройку в соответсвии

с содержательной информацией о задаче. Как правило, такие системы представляют собой комбинацию существующих систем, например Isabelle [] или система предложенная в [problem-oriented application of ATP]. Coq предлагает использование так называемых «тактик» (соединение нескольких типовых шагов вывода в один шаг). Интерактивные системы, хотя и не могут в полной мере быть автоматическими, всё же они интересны тем что используют философию тесного взаимодействия человека и компьютера.

Конечно, некоторые системы могут быть причислены сразу к нескольким классам.

Авторами самых передовых пруверов неоднократно высказывалось что сложность разработки начинает достигать своего предела. Внедряемые методики крайне сложны в реализации, в совмещении с другими методиками, портят расширяемость систему, и вносят абсолютное запутывание в то что происходит внтури системы во время работы. Так, например, в работе [**BTPstickel**], автор M. Stickel приводит главу с названием в переводе на русский язык «Индексирование, необходимое зло», при этом сам Стикель является автором одной из очень популярной и применяемой в самых передовых пруверах методики индексироваия путями (path indexing).

В работах [**Eprover**] указывается на то что их система более интеллектна по сравнению с другими системами. Такая интеллектность обсуловлена возможностью более гибкого подключения дополнительных эвристик для решения задач определенного класса.

Отсюда можно сказать что в целом действительно в области АДТ есть необходимость в новых методах либо в развитии старых в том направлении чтоб привлекать возможности человека а так же гибкости систмы для варьирования своего поведения в зависимости от решаемой задачи.

Отметим что существуют теоремы, которые имеют сколь угодно большой минимальный вывод, т.е., даже если прувер сумеет перебрать все варианты доказательства до определенной глубины, то он всё равно не докажет теорему лежащую вне этого пространства поиска. Отсюда, исследование любых методов, позволяющих расширить возможное пространство поиска

(глубину вывода), является актуальным. С нашей точки зрения исчисление ПО-формул как раз даёт фундаментальное сокращение шагов вывода, в силу крупноблочности правила вывода.

С другой стороны, такое отодвижение границ даёт лишь потенциальную возможность для доказательства более широкого класса формул. Полный перебор с возрастанием глубины вывода отнимет очень много ресурсов. Поэтому так же актуальным является и возможная интеллектуализация процедуры поиска вывода, т.е., дополнительные эвристики о задаче, позволяющие двигаться в предположительно верном напарвлении поиска.

Многие современные системы АДТ разрабатываются уже много лет (есть примеры 30 летнего опыта), в них внедрено огромное количество методик. Тем не менее эти методики как правило не носят интеллектный характер, а направлены лшиь на эффективную обработку данных (например, индексироваие), сокращение потребляемой памяти и т.д. Либо предложен некоторый ограниченный ряд стратегий общего характера.

Кроме того в силу интеллектности и свойства интерпретируемости вывода в исчислении ПО-формул, было бы интересно разработать методики преобразования формального вывода к виду, пригодному для понимания человеку.

1.2. Теоретический базис работы

В основе разрабатываемой системы лежит исчисление ПО-формул.

Исчисление ПОФ $\boldsymbol{JF}$ есть тройка $\langle \boldsymbol{LF}, Ax\boldsymbol{JF}, \boldsymbol{\omega} \rangle$, где $\boldsymbol{LF}$ — язык ПОФ, $Ax\boldsymbol{JF}$ — 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 *конюнкт* 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 ПОФ если оно построено только по правилам Определения 1.

Переменые 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 ПОФ как семантику соответствующих формул языка предикатов первого порядока.

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

Любая ПОФ очевидно имеет структуру дерева. Таким образом, для удобства читаемлсти мы будем представлять их как древовидные струк-

туры а также пользоваться соответсвующей терминологие: узел, корень, ветвь, лист и т.д.

Если ПОФ F начинается с $\forall\colon \boldsymbol{True}$ узла, и каждый лист F является $\exists$-узлом, то F называется ПОФ в *канонической форме*. Очевидно что любая ПОФ 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 ПОФ начинающаяся с $\forall\colon \boldsymbol{True}$.

2. Если F есть $\exists$-формула, тогда $\forall\colon \boldsymbol{True}\,(F)$ есть ПОФ 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.

В дальнейшем, если не оговорено обратного, будем рассматривать только ПОФы в каноническом виде.

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

Пример 1 Рассмотрим формулу языка FOL

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

Образ F^{PCF} формулы F в языке ПОФ 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 ПОФ $\boldsymbol{JF}$ 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 исчислиении ПОФ для того что бы доказать F мы будем пытаться опровергнуть её отрицание, поэтому аксиомы выбраны тождественно равные лжи. Таким образом процесс вывода в исчислении ПОФ 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 универс основанный на $\exists$-переменных из $\bar{x}$, константных и функциональных символах которые встречаются в соответствующей базе.

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 неоторых переменных в каждой подформуле, выражение $\boldsymbol{\omega}F$ будет удовлетворять всем условиям правильно-построенных ПОФ.

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

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

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

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:

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

На третем шаге вывода первая база может быть опровергнута ответом

$\{x \to y_1; y \to a\}$ на целевой вопрос Q_3. Опровергнутая база (фодформула) 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 преобретает следующий вид:

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

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

Опровержение закончено поскольку все базы опровергнуты.

Корректность и полната доказаны в.

1.2.3. Особенности ПО-формул

В литературе выделяются различные положительные стороны исчисления ПО-формул. Выделим те, которые на наш взгляд являются наиболее подходящими для данной работы.

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 кванторов существования). Процедура сколемизации приводит к увеличению сложности термов а значит ив сей формулы. Кроме того данная особенность делает исчисление боеле 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 $\exists x(A\&\sqcup)$ [**Bourbaki**], [**ICDS2000**].

5. Исчисление ПО-формул содержит только одно правило вывода и это свойство (как и в случае МР) делает исчисление пболее машинно-ориентированным. Кроме того правило выводя является крупноблочным, что лучше сказывается на человеко-оринтированности.

6. Процедура ЛВ фокусируется в ближайшей окрестности корня ПО-формулы, благодяра особенностям 1, 2.

7. ЛВ может быть представлен в терминах *вопросно-ответной* процедуры, а не в технических терминах формального вывода (в терминах логчиеских 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 конкретнъми 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. Такая интерпретируемость доказательства довольно важна с точки зрения человеко-машинных приложений. Таким образом, как говорилос выше, исчисление ПО-формул не только машинно-ориентированное, но и человеко-ориентированное.

11. Семантика исчисления ПО-формул может быть изменена без какой либо модификации аксиом или правла 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 дп. Примеры использования таких семантик представлены в [**ICDS2000**].

1.3. Формальная информация о диссертации

Объектом исследования разработка эффективных алгоритмов для автоматического доказательства теорем в исчислении ПО-формул. Под эффективностбю понимается: время решения задач; количество шагов логического вывода; ширина класса решаемых задач.

Предметом исследования являются методы повышения производительности логического вывода, разработанные и адаптированные с других систем АДТ.

Методика исследования. Использование исчисления ПО-формул как базиса для АДТ; Использование языка D для программирования системы; Анализ других систем АДТ на предмет применения их методов.

Цель диссертационной работы: Разработка высокопроизводительной человеко- машинно-ориентированной программной системы для автоматического доказательства теорем в исчислении позитивно-образованнх формул.

Основные задачи диссертационной работы. Для достижения описанной выше цели решаются следующие задачи:

1. Исследование языка и исчисления ПО-формул, анализ его сильных и слабых сторон;
2. Разработка эффективных структур данных представления ПО-формул формул в памяти компьютера;
3. Разработка алгоритмов обработки ПО-формул для организации автоматического логического ввода;
4. Адаптация существующих методов АДТ для исчисления ПО-форул;

5. Исследование вопросов эффективного вывода с использованием предиката равенства;

6. Реализация инфраструктуры для участия человека в процессе логического вывода;

7. Апробация инструментальных средств в решнии задач разного рода.

Научная новизна, практическая значимость и апробация полученных результатов Разработаны новые и адаптированы существующие алгоритмы, реализующие метод АДТ в исчислении ПО-формул. Использование данных алгоритмов позволило реализовать полноценную программную систему АДТ для исчисления ПО-формул. В результате проведенного исследования получены следующие новые научные результаты:

1. Стратегия логического вывода ПО-формул с неограниченными переменными;

2. Стратегия k,m-условия;

3. Стратегии разделения памяти для системы АДТ ПО-формул;

4. Индексирование термов для системы АДТ ПО-формул;

5. Стратегии параллельного логического вывода для системы АДТ ПО-формул;

6. Выявлены слабые стороны исчисления ПО-формул;

Практическая значимость представляется следующими основными результатами:

1. Выделены классы задач, на которых разработанная система ведёт себя более эффективно чем самые производительные современные систмы АДТ;

2. Налажено взаимодействие с TPTP;

3. Расширена область применения систем АДТ (Леса?)

Кроме того немаловажным результатом является возможность использования системы АДТ базирующейся на раннее не использованном исчислении ПО-формул (новый взгляд на задачи).

Работы и исследования, проведенные в рамках диссертации, выполнены в ФГБОУ ВПО «Иркутский государственный университет» и в Институте динамики систем и теории управления СО РАН.

Исследования поддержаны также грантами:

1. РФФИ были какие-то?

2. Университетский кластер.

Разработанные инструментальные средства внедрены в учебный процесс вузов города Иркутска, в частности, в Институте математики, экономики и информатики Иркутского государственного университета (ИМЭИ ИГУ). Разработанановый вариант курса “Технологии разработки программного обеспечения” на кафедре информационных технологий ИМЭИ ИГУ.

Инструментальные средства также внедрены...

Представление работы. Материалы работы докладывались на

- Международной конференции «Мальцевские чтения», г.Новосибирск, 2009 г.;
- Семинаре ИДСТУ СО РАН «Ляпуновские чтения», ИДСТУ СО РАН, г. Иркутск, 2009 г.;
- Всероссийской конференции молодых ученых «Математическое моделирование и информационные технологии», г. Иркутск, 2010 г.;
- Международной конференции «Облачные вычисления. Образование. Исследования. Разработки», г.Москва 2010 г.;

- Международном симпозиуме по компьютерным наукам в России. Семинар «Семантика, спецификация и верификация программ: теория и приложения», г.Казань, 2010 г.;
- 4-ая Всероссийская конференция «Винеровские чтения», г.Иркутск, 2011г.
- 34-ый международный симпозиум «MIPRO», г.Опатия, Хорватия, 2011г.
- 4-ая Всероссийская мультиконференция по проблемам управления, с. Дивноморское, 2011г.
- КИИ-2012

Публикации по теме диссертации. По теме диссертации опубликовано 6 печатные работы cite[] (по списку литературы), пять статей cite[] опубликована в журналах из перечня рецензируемых научных журналов и изданий ВАК, в которых публикуются научные результаты диссертации на соискание ученой степени доктора и кандидата наук.

Личный вклад автора. Все представленные результаты получены лично автором или в соавторстве с научным руководителем и Давыдовым А.В. Автором лично разработаны:

1. Механизмы логического вывода в исичлсении ПО-формул (структуры данных и их обработка);
2. БОЛЬШЕ НИЧЕГО ЛОЛОЛО

Из печатных работ, опубликованных диссертантом в соавторстве, в текст глав 2,3 и 4 диссертации вошли только те результаты, которые содержат непосредственный определяющий творческий вклад автора диссертации на этапах проектирования и разработки программного обеспечения. В перечисленных публикациях все результаты, связанные с вопросами реализации и использования программной системы принадлежат автору.

Соответствие паспорту специальности 05.13.11 Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей. Область исследования 5. Программные системы символьных вычислений; 8. Параллельное

Структура и объем диссертации. Диссертация состоит из четырех глав, первая из которых — вводная, заключения, списка литературы и приложения. Основной текст изложен на 62 страницах машинописного текста, полный объем диссертации 63 страниц. В работе содержится 20 рисунков. Список литературы содержит 50 наименований.

Глава 2 посвящена теоретическому базису программной системы, в частности описанию языка и исчисления ПО-формул, полезным свойствам ПО-формализма а также некоторым проблемам этого формализма, которые (в том числе) необходимо преодолеть для достижения поставленной цели.

Глава 3 посвящена аспектам реализации и использования системы.

В главе 4 представлены методы и результаты тестирования системы.

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 2. Алгоритмическое обеспечение и адаптация алгоритмов

2.1. Предпосылки разрабатываемых алгоритмов

Для того что бы определить алгоритмы необходимо выявить основные полезные свойства и проблемные места ПО-формализма, для того что бы алгоритмы адекватно использовали возможности формализма. Полезные свойства представлены во введении и являются общепризнанными.

2.1.1. Проблемы формализма ПО-формул

Укажем на некоторые негативные стороны ПО-формализма, в частичном разрешении которых он нуждается.

1. Поиск ответов на вопросы с открытыми переменными требует выбора подставляемого терма для данной переменной из эрбранова универса, который в общем случае (при наличии функциональных символов) является бесконечным множеством (счетным множеством). Какой именно терм необходимо выбрать — изначально неизвестно.

   Сделаем несколько замечаний. Во-первых, при решении прикладных задач, переменные связанные квантором всеобщности ограничиваются типовым условием, а значит появление открытых переменных в данном случае следует рассматривать как некую аномалию в следствии неверной формализации задачи. Это значит, что решение данной проблемы лежит вне прикладной области, и необходимо для решения общематематических теорем (например из библиотеки TPTP). Во-вторых, есть соблазн уйти от функциональных символов (как это скромно сделано в [**Vas_ICDS**]), но наличие функциональных символов есть основа

сложных задач. В-третьих, в [**Vas_ICDS**] предложена идея стратегии решения данной проблемы — стратегия отсроченного присваивания (СОП), заключающаяся в том что изначально для открытой переменной выбирается неопределенный эрбранов элемент, а позднее он постепенно доопределяется. Подробное описание стратегии и её реализация представлены в главе 3. Данная стратегия конфликтует с другими стратегиями. Однако анализ решаемых задач показывает что в прикладных задачах (как-то их иначе наверное надо назвать) нет нужды использовать СОП ибо нет открытых переменных, а в общих задачах сложнее найти какие-то особенности которые применяются другими стратегиями, а значит конфликт частично разрешается просто разграничением областей применения системы.

2. Язык ПО-формул позиционируется как «достаточно однородноый, но в то же время хорошо структурированный» и хорошо усваивающий эвристики. Стоит заметить, что однородность ПО-формул хуже чем, например, однородность дизъюнктов, в силу разнородных сущностей структуры формулы: база, вопросы, консеквенты вопросов. Разрешение данного вопроса требует применения специальных методов доступа к данным неоднородным частям.

3. Хотя изначально представление формулы ИП в языке ПО-формул более компактно чем КНФ, дальнейшее проведене правила вывода может привести к бОльшему усложнению структуры формулы (при наличии дизъюнктивного ветвления), чем при применении правила вывода в МР. Таким образом через некоторе количество шагов ввода размер ПО-формулы может оказаться в разы больше чем размер КНФ. Но эта проблема почти полностью устранима чисто техническими способами, а именно описанными далее методами разеления данных, а также ограничивающей страгией (которую пользователь выбирает сам).

2.1.2. Подходы

Поскольу 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 подстановок. Кроме того существуют общетехнические методы независящие от конкретного формализма, например параллельные схемы алгоритмов, экономия потребляемой памяти, индексирование данных.

Таким образом в работе предполагается адаптация некоторых существующих алгоритмов, используемых в современных и самых эфективных системах АДТ. Адаптация предполагает, во-первых, учитывание особенностей ПО-формализма - как положительных так и проблемных, во-вторых, совмещение (синхронизация) алгоритмов, поскольку некоторы из них конфликтуют при прямой реализации.

2.2. Алгоритмы и структуры данных

2.2.1. Общие структуры данных

Типы данных. Язык представления формул имеет следующие типы данных: ATOM (атомарный символ), FUNCTION (функциональный символ), CONSTANT (константный символ), AVARIABLE (универсальная переменная), EVARIABLE (экзистенциальная переменная), UHE (Неопределённый Эрбрановский Элемент), INTEGER (целочисленный символ), FLOAT (нецелочисленный символ), STRING (строковый символ). Данные типы определяются перечислением:

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

Символ (Symbol). Символ - буква из алфавита используемого языка. Структура Symbol содержит строковое представление символа, тип данных, арность. Символ индентифицируется по его адресу в памяти.

Обобщённый терм (GTerm). Такое название взято из [**NNN**]. Смысл

заключается в том что для представления как терма так и атома используется одна структура данных, поскольку и первые и вторые имеют древовидную структуру. Для правильной обработки соотвествующих обобщенных термов используются вышеописанные типы данных. Терм представляется классической древовидной структурой каждый узел которой содержит символ верхнего уровня и массив ссылок на дочерие узлы. Универсальная переменная (AVARIABLE) и неопределённый эрбрановский элемент (UHE) используют один дочерний узел как ссылку на терм, который подставляется вместо данного. Если ссылка не NULL значит переменная связана с некоторым термом. В противном случае переменная свободня для подстановки. Для того что бы корректно обрабатывать перменные, задан метод getValue() который возвращает значение типа GTerm. Если переменная не связана то значение getValue() совпадает со ссылкой на эту переменную, в противном случае возвращается ссылка на терм с которым связана переменная.

Чанк. Маркированный список. Список разделённый на подсписки любой длины (в том числе нулевой). Для каждого элемента списка можно определить к какому подсписку он относится. Далее будет показано что в каждом чанке хранится информация о текущем шаге выводе, а весь список в целом хранит всю информацию о формуле. Т.е. формула это список, а чанк это подформула полученная на определённом шаге вывода. На рисунке представлена структура чанкового списка.

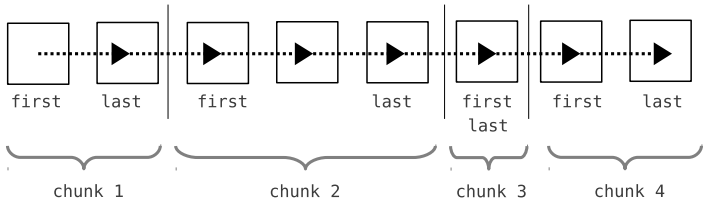

Рис. 2.1. Чанковый список

На следующем рисунке показан случай когда один из чанков пуст. Стоп.

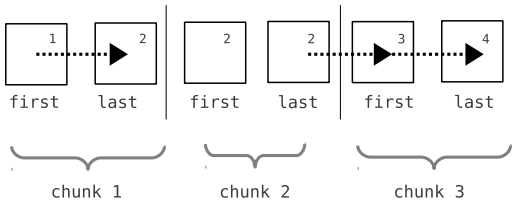

Рис. 2.2. Чанковый список

2.2.2. Дерево состояний вывода

Одним из основополагающих методов в разработанной системе АДТ является дерево состояний вывода (ДСВ), состоящее из структур данных, характеризующих логический вывод и ряда методов, обрабатывающих данные структуры. Основная цель данного подхода заключается в том что бы строго зафиксировать все события произошедшие на каждом шагу логического вывода, например что бы точно определить какие данные были выведены на каком шагу вывода. Такая фиксация событий позволит:

1. Более глубоко анализировать процесс ЛВ, а значит эффективно внедрять эвристики.

2. Производить откаты в выводе (backtracking).

3. Реализовать один из подходов разделения данных.

4. Производить эффективное освобождение памяти.

Для начала, кратко объясним идею ДСВ. При классическом подходе после каждого ответа на вопрос к первоначальной формуле добавляется консеквент вопроса на который производился ответ (в базу добавляются соответствующие элементы узлов непосредственно следующих за вопросом; к вопросам добавляются новые вопросы, если они есть), а в случае дизъюнктивного ветвления соответствующая база расщепляется, не теряя никаких данных. Таким образом, формула монотонно увеличивается, при этом сохраняя свою структуру. ДСВ будем использовать для того что бы

иметь полную информацию о текущей формуле, для возможности отката в выводе и подробного наблюдения за ним. Опишем ДСВ формально.

Дерево состояний вывода есть такое дерево, которое обладает следующими свойствами: корень дерева есть одна из базовых подформул первоначальной формулы; все остальные узлы есть добавляемые консеквенты (с примененным к ним подстановками-ответами и разыменованием переменных). Если приводить в пример определение правила вывода, то корень дерева это база E, а узлы это E2theta. Таким образом если происходит расщепление то в данном узле появляется ветвление. Отсюда можно говорить что каждая базовая подформула в формуле характеризуется соответствующим путём от листа до корня.

Как видно, каждый узел содержит самодостаточную информацию и для того чтоб производить откаты назад, достаточно 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 ближайшего ветвления, поскольку оставшаяся часть пути всё ещё используется для представления другой базы. Количество узлов равно текущему количеству баз. Если дерево пусто, значит первоначальная база опровергнута. Та как изначальная формализация задачи в языке ПО-формул может содержать несколько базовых подформул, то для каждой из этих подформул строится своё ДСВ.

Для практических нужд узел ДСВ содержит некоторую системную информацию:

1. Множество атомов, понимаемых как часть базовых, отсюда каждый базовый конъюнкт характеризуется объединением всех множеств от данного узла до корня.

2. Список ссылок на вопросы к базе.

3. Для каждого вопроса определяется чанк ответов для вопроса в данный момент.

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

5. Чанк уже отработанных вопросов.

Узлы содержат ещё различную информацию, которая будет описана ниже в описании стратегий вывода.

За счёт чанков получается разграничивать данные полученные на каждом шаге, с другой стороны на каждом шаге (в каждом узле) доступные все собранные до этого данные.

ДСВ для формулы из примера представлено на рисунке Рис. 2.3.. Корнем ДСВ является первоначальная 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 являются соответствующими консеквентами для 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. Когда 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.

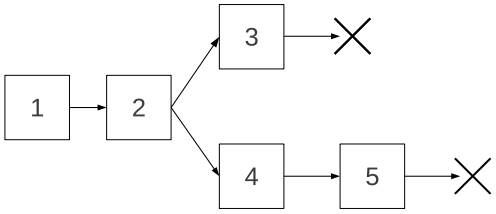

Рис. 2.3. Дерево состояний вывода для формулы из примера 2.

На следующем рисунке представлено ДСВ с точки зрения его связи с чанками.

С точки зрения реализации ДСВ растёт от листо к корню.

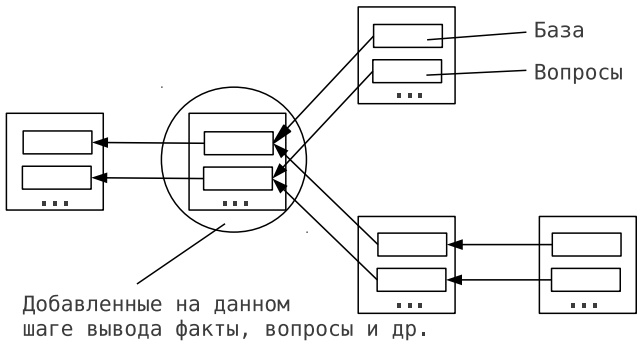

Рис. 2.4. ДСВ.

2.2.3. Супервизор

Наблюдает глобально за всем деревом. Содержит информацию об ограничении ресурсов, эвристики и др. В супервизоре находится список всех текущих листов дерева. Большинство стратегий реализовано на уровне супервизора, посколько иногда приходится читывать информацию о всей совокупности предшествующего или последующего вывода.

Наличие такого субъекта управления обсуловлено тем что некоторые частные события в процессе логического вывода могут повлиять на него в целом.

2.2.4. Стратегия ограниченных ресурсов.

Стратегия ограниченных ресурсов есть в Вампире. У нас подобная стратегия напрямую зависит от текущего ДСВ и описывается так: ветка ДСВ строится до тех пор, пока не наступит ограничение разрешенной для использования памяти, либо пока не истечет определенное время, либо пока не будет произведено определенное количество шагов вывода. Если наступил предел использования ресурсов, то необходимо произвести откат назад и попробовать другие ответы, т.е. другие варианты построения дерева.

2.2.5. Разделение данных

Логический вывод как правило связан с выводом новой информации. Например, в МР выводятся новые дизъюнкты до тех пор пока не выведется пустой, а в методе доказательства ПО-формул производится насыщение баз фактами до тех пор пока они не станут противоречивыми. Поскольку сложность формул может быть сколько угодно большой и даже минимальный вывод может иметь сколько угодно большую длину, имеет место быть проблема увеличения затрачиваемых ресурсов на всеразростающиеся формулы. Опыт показывает [ссылка] что автоматический вывод довольно быстро занимает всю имеющуюся в распоряжении память и далее процесс вывода требует регулярное удаление излишков. Таким образом проблема экономии памяти довольно важна, особенно с учётом увеличения сложности задач. Для экономии памяти используются, во-первых проектирование компактных структур данных, во-вторых методы раздления общих участков памяти (data sharing). В случае логических языков и конкретно языка ПО-формул метод раздлеения данных довольно актуален.

Исходя из некоторых общих особенностей представления языков первого порядка и представления ПО-формул, нами выделено четыре вида разделения данных.

Агрессивное разделение термов. 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 достаточно сильно экономить оперативную память при ограниченных ресурсах, однако требует дополнительное процессорное время на вычисление общих подтермов. Такой метод является общеупотребимым в системах АДТ, классически варианты реализации представлены в [].

Мягкое разделение термов. Отличается от агрессивного намного большей эффективностью в смысле потребления процессорных ресурсов, но меньшей эффективностью с точки зрения экономии памяти, поскольку

разделяет только часть общих подтермов. Опишем данный метод более подробно. Исходя из определения 3 применение правила вывода ω законно 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 память. θ же является отображением переменных вопроса A в элементы эрбранова универса. В дальнейшем при применении правила вывода θ применяется ко всему консеквенту вопроса и данный консевент добавляется к формуле. Однако правая часть подстановки уже имеется в памяти, в силу того что основана она на B. Исходя из этого достаточно использовать ссылки на уже имеющуюся память используемую для правых частей подстановки в тех частях консеквента где эта подстановка применяется. На рисунке детально представлена данная ситуация.

Разделение базовых подформул. ПО-формулы, в которых производится ответ на вопрос имеющий вопрос с дизъюнктивным ветвлением, расщепляются на несколько новых базовых подформул. Количество новых базовых подформул совпадает с количеством непосредственных наследников в консеквенте вопроса. В прямом варианте такое расщепление требует копирования предыдущего состояния формулы несколько раз, такое копирование естественно приводит к большим затратам памяти и процессорному времени предназначеннго для копирования. Решение данного вопроса могло бы лежать в плоскости агрессивного разделения термов, однако такого разделения может быть недостаточно. Если формула предполагает достаточно сильное ветлвение, сохраняется проблема наличия множества ссылок на разделяемые атомы баз. Поскольку расщепление предполагает разделение баз, то имеет смысл разделять и эти ссылки. Более подробно ситуация рассмотрена на рисунке.

Таким образом в системе кроме разделения термов используется ещё и разделение ссылок на эти термы, благодаря ДСВ.

Разделение переменных и неопределенных эрбрановских элементов. Предназначено для быстрого применения подстановки ко всей формуле, т.к. все одноименные переменные в формуле представляют собой один участок в памяти. О неопределенных эрбрановскиъ элементах сказано ниже. Все одноименные переменные на протяжении любого основного пути формулы [**dissChe**] 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.

Веса подформул. Кроме непосредственно экономии потребляемой памяти, реализована возможность сдерживания разрастания формулы. Для этого из возможных ответов на вопрос выбирает тот который приводит к формуле наименьшего веса. Под весом понимается количество узлов в дереве представлющем терм или формулу.

2.2.6. Индексирование данных

Из определния стратегии ясно, что шаг вывода зависит от некоторых **критериев**, которым должны удовлетворять определенные части ПО-формулы. Например, допустимы следующие критерии: Наличие в базе заданного терма, наличие вопроса с заданными свойствами, количество вхождений терма в конъюнкт и т.д. Для определения истинности данных критериев необходимо проводить поиск и ианализ данных в формуле. С каждым шагом вывода формула в общем случае увеличивается (причем в некоторых сулчаях довольно сильно), и может настать момент когда поиск тех или иных данных будет затруднен.

Индексирование термов В информационных технологиях подобные проблемы разрешены с помощью методов индексирования данных (широкоприменяемых в базах данных [**Ulman**]). Подобно тому как в библиотеке

книги проиндексированы первыми буквами своих названий а также именами авторов. Такие подходы используются и в реляционных БД.

В нашем случае основнй объект индесирования есть терм, который является древовидной структурой, индексирование которой в реляционных БД неэффективно [есть ссылка на это]. Поэтому следует использовать иные подходы.

Индексирование термов — хорошо исследованный вопрос, как в рамках определенных систем АДТ, так и абстрактно. В частности по данной теме можно почитать [дискриминационное дерево и пути, подстановочное дерево, и книжку Графа про индексирование]. Перечисленные методы, позволяют эффективно находить в базе термов те термы, которые удовлетворяют определенным критериям: являются равными данному (query term), являются его примерами, обобщениями (generalization) и унификациями. Для наших нужд не требуется обобщение и классическая унификация, но дополнительно необходимы НЭЭ Унификация, критерий наличия термов включающих заданный подтерм, и различные количественные критерии: вес, глубина, арность. С учетом перечисленных требований а также того факта, что как правило в базе находятся основные термы, мы в качестве основы выбрали методику индексирования путями [].

Кратко опишем её суть, как она описана в []. Для каждого атома составляется список так называемых путей. [Надо привести формальное опредление пути из [макКьюн]]. Например, атом $A(g(x,c),e)$ представляется в виде путей A, $A1g$, $A1g1x$, $A1g2c$, $A2e$. Каждый из этих путей содержит указатель на соответствующй атом. Сами пути хранятся в отсортированном виде (в дереве).

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

Заметим, что методы индексирования термов на сегодняшний день используются во многих известных системах АДТ [Vampire -улучшенное индексирование путями, Otter -дискр.дерево и пути, E, EQP, SPASS -дерево подстановок].

Кроме базы термов, в индексировании нуждаются множества вопросов и множества ответных подстановок.

Индексирование других частей формулы Теперь упишем иные нужды индексирования. Предположим что у нас имеется некоторая стратегия для решения некоторого класса задач. Стратегия оперирует вопросами с определенными свойствами (т.е. вопросу сопоставляется содержательная информация), которые присущи всему классу задач (т.е. все задачи объединяет наличие определенных вопросов). Естественно было бы неплохо как-то закрепить эти вопросы что бы упростить в дальнейшем к ним доступ и каждый раз не проверять все вопросы на наличие определенного свойства. Для этого достаточно сделать словарь (map) в котором каждому описанию вопроса соответствовал бы указатель на данный вопрос. (Конечно можно в стратегии пользоваться лишь номером вопроса, но тогда теряется гибкость, ведь пользователю необходимо будет ставить вопросы всегда на одно и тоже место.) Данный вид индексирования (весьма простого) может выполняться как на этапе компиляции (тогда в компиляции прувера будет задействован не только стратегия но и сама задача), так и при первичном обрещнии к вопросу (тогда нет необходимости в компилировании самой задачи).

2.2.7. Неограниченные переменные

Проблема неограниченных переменных описана выше. Для её решения предложено несколько подходов.

Ленивая конкретизация Суть заключается в следующем: для открытой переменной не выбирается конкретный элемент эрбранова универса, а в место этого она заменяется на неопределенный эрбрановский элемент (НЭЭ), который в дальнейшем исходя из нужд вывода постепенно доопределяется до основного терма, либо в некоторых ситуациях так и остается недоопределенным. По своей природе НЭЭ схож с переменной в том смысле что он изменяем, однако все изменения касающиеся НЭЭ должны быть направлены на конкретизирование НЭЭ, т.е. НЭЭ может быть заменен либо на некий терм (возможно тоже содержащий НЭЭ) либо на другой НЭЭ, но не на переменную. Однако для переменной НЭЭ считается как терм, и можно производить замену на него.

$$\forall\colon \boldsymbol{True} - \exists\colon \boldsymbol{True} - \begin{cases} \forall x\colon \boldsymbol{True} - \exists\colon A(x) \\ \forall x\colon A(f(x)) - \exists\colon B(f(x)) \\ \forall\colon B(f(a)) - \exists\colon \boldsymbol{False} \end{cases} \tag{2..1}$$

На первом шаге вывода имеется ответ $\{x \to h_1\}$ на первый вопрос, x является неограниченной переменной, h_1 это неопределённый эрбрановский элемент (НЭЭ). После первого шага атом $A(h_1)$ Добавляется в базу. На втором шаге вывода имеется ответ $\{x \to h_2\}$ на второй вопрос, и h_1 конкретизируется до $f(h_2)$, после второго шага $B(f(h_2))$ добавляется в базу. Наконец, на третем шаге имеется тривиальный ответ на третий вопрос и h_2 доопределяется до a.

Однако существуют особые ситуации. Рассмотрим следующий пример:

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

Данная формула имеет вывод, например такой: отвечаем на первый вопрос с 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 имеет несколько

вариантов ответа, но 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 целевой вопрос имеет ответ. Заметим что на первый вопрос необходимо обязательно выбирать те подстановки котоыре содержат $y \to e$ и $y \to f(e)$ необходимые для ответа на второй и третий вопросы соответственно. Также формула устроена таким образом что первый и второй вопросы всегда имеют новые ответы (надеюсь это очевидно).

Теперь рассмотрим работу стратегии отсроченного присваивания: ответ на первый вопрос будет следующего вида $\{x \to e, y \to h_1\}$, где h_1 есть НЭЭ, и в базу попадают следующие факты $S(h_1), M(f(e)), T(e)$; ответ на второй вопрос есть $\{x \to e, h_1 \to e\}$ в котором h_1 доопределяется до e и соответсвенно находящийся в базе факт $S(h_1)$ доопределяется до $S(e)$; с этого момента начинается зацикливание, поскольку целевой вопрос не имеет ответа, вновь отвечаем на первывй вопрос и вновь в базу попадает факт $S(h_1)$, который при ответе на второй вопрос вновь доопределится до $S(e)$ и т.д. Однако если пропустить ответ на второй вопрос, то при ответе на целевой, факт $S(h_1)$ доопределится до $S(f(e))$ и на этом вывод закончится.

В общем виде проблема заключается в том, что НЭЭ доопределяется при поиске ответов на определенный вопрос, раньше чем это необходимо при ответе на другой вопрос. То есть доопределяется там где этого уже не надо.

Из подобных примеров видно, что СОП не может использоваться на прямую. Необходимы какие-то дополнительные средства обеспечения логического вывода.

Первый вариант 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 которых будет использоваться для второго вопроса, а другой для целевого. Такой вариант вполне приемлем и нами предпо-

лагается что именно он и будет чаще всего использоваться для решения задач. Но что делать, если дополнительные знания о задаче не позволяют сформулировать правильную стратегию? Необходимо предоставить хотя бы принципиальную возможность вывода (что пока необеспечно).

Второй вариант Мы видим следующий путь решения данной проблемы в общем случае (т.е. без использования дополнительных стратегий): не доопределять НЭЭ на месте (как это сделано выше), а **порождать** новые выржаения, полученные из исходных, заменой в них НЭЭ на доопределение. То есть если в базе содерджится факт $S(h_1)$ и h_1 необходимо доопределить до некотого терма t, то в базу добавляется новый факт $S(t)$, порожденный от $S(h_1)$, при этом сам $S(h_1)$ сохраняется. При определенных условиях также может возникнуть ситуация когда будут порождаться не только факты, но и целые подформулы. Содержательно такой подход означает следующее: можно считать что для вопроса с откртыми переменными применяется вся совопкупность возможных ответов (с использованием всех элементов эрбрановского универса), но так как технически это нереализуемо (из-за бесконечности эрбановского универса), используется техника ленивых вычислений. В уме мы думаем что применили все ответы, а на деле используем только необходимые.

Данный подход имеет и определенные недостатки. Связано с тем если появится вопрос с НЭЭ, то придется порождать новые вопросы, а если будет диз.ветвление, то ещё хуже. Плюс во время унификации вылазят всякие проблемы. Короче об этом надо упомянуть но сильно не выпячивать, и оправдаться тем что мы больше нацелены на использование миниязычка, а эта стратегия дана лишь для обеспечения полноты вывода. Но эта стратегия вполне эффективна если НЭЭ не будут появляться новые вопросы с НЭЭ и в диз.ветвлении их не будет...

Отметим, что стратегии основанные на использовании НЭЭ стоят особняком для многих других стратегий. Во-первых из-за этой стратегии нарушается независимость базовых подформул, поскольку один и тот же НЭЭ может быть в разных подформулах, то доопределение одного из них в од-

ной подформуле автоматически приводит к доопределению его в другой подформуле, это в частности противоречит параллельным стратегиям и противоречит некоторым эвристическим фишкам.

Стратегия фильтрации эрбрановского универса Ещё одним вариантом решения проблемы открытых переменных является стратегия фильтрации ЭУ. Данная стратегия позволяет избавиться от проблем вышеперечисленных, но несколько расширяет пространство возможных ответов. Суть данной стратегии заключается в следующем. Атомы консеквентов, которые (атомы) содержат неограниченные переменные вопросов, должны унифицироваться с одноимёнными атомами из всех других частей всей базовой подформулы, в противном случае атом попавший в базу не будет удовлетворять условие применения правила вывода, поскольку никода не выполнится процедура матчинга при поиске ответов. Будем использовать даное свойство в стратегии. В качестве ответа для неограниченных переменных используются основные примеры унификации. Такой подход позволяет заранее сузить пространство ЭУ, фильтруя заведомо бесполезные ответы.

Для ясности рассмотрим пример.

2.2.8. Стратегия km-условия

Данная стратегия формулируется следующим образом. Некоторый ответ применяется, если за последующие k шагов произойдет заданное событие m раз. Пользуясь терминологией ДСВ это означает что для данного узла дерева применяется ответ в случае если построенное в результате дальнейшего вывода поддерево корнящееся с этого узла не привысит глубину k и до этого момента произойдет m раз событие.

Мы преддлагаем три спецификации данной стратегии.

km-опровержение. Ответ выбирается в случае если за последующие k шагов будет опровергнуто m баз. Подобная стратегия первоначально предложена в [**ICDS2000**] и реализована в КВАНТе Черкашина. В дан-

ной системе эта стратегия расширена вторым параметром m. Она позволяет сдержвать разростание пространства поиска вывода, т.е., сдерживает излишнее ветвление ДСВ, что в некоторых случаях приводит к многократному усложнению вывода.

km-конкретизация. Если за последющие k шагов будет доопределено m НЭЭ. Эта стратегия также направлена на то чтоб ограничить сложность представляемой формулы. Недоопределенный НЭЭ тащит много информации за собой и много условий, что на уровне реализации влияет негативно, поэтому чем больше и быстрее НЭЭ будут доопределены нужным образом, тем лучше.

Особенности реализации Основной для реализации данной стратегии являются описанные выши ДСВ и супервизор. Параметры k,m и собтсвенно условие закладывается в супервизоре и привязывается к определённом узлу ДСВ. Через каждый следующий шаг вывода, супервизор проверяет все условия. И в положительном случае производится откат назад, т.е. последовательное удаление узлов дерево в обратном порядке (от листа в напарвлении корня). Перед удалением каждого узла производится разконкретизация НЭЭ полученных на этих узлах и возврат использованных вопросов в стадию активных (т.е. возможных для применения).

2.2.9. Кэширование результатов

Учитывая множественность возможных ответов, откатов 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 ответты в итоге могут совпадать. Все примененные ответы сохраняются, причем хранятся они как и база в пчанках, размазанных по всему пути от листа до узла. Это

позволяет делать коорректные откаты с учетом сохранения информации о примененных ответах.

2.2.10. Параллельные стратегии

Полезно сполна использовать все вычислительные ресурсы. Многие современные ВС обладают бОльшим чем 1 вычислительным элементом (процессором).

Предлагаем следующие стратегии для распараллеливания процесса логического вывода.

Первая стратегия В случае если вопрос имеет дизъюнктивное ветвление, то после ответа на этот вопрос, формула расщепляется и трансформируется в формулу с б'ольшим количеством баз. Таким образом, количество базовых подформул может заметно увеличиваться. Кроме того, исходная формализация задачи в языке по-формул может содержать более одной базовой подформулы. Для того, что бы показать, что исходная формула противоречива, необходимо опровергнуть каждую из баз. Специфика исчисления ПО-формул, позволяет рассматривать данные базы независимо друг от друга (естественный ИЛИ-параллелизм, следующий из того что в базах находятся лишь основные термы), а значит, процедура опровержения каждой базы может выполняться в отдельном вычислительном процессе. Таким образом, первая стратегия, реализуемая в виде параллельной схемы алгоритмов, формулируется следующим образом: каждая базовая подформула опровергается независимо, а значит параллельно. Для нужд данной стратегии как раз и реализовано жесткое копирование (следующая глава), что бы полностью скопировать базовую подформулу и обрабатывать её в отдельном процессе независимо (не разделяя память).

Вторая стратегия 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.

Третья стратегия Теперь рассмотрим процедуру поиска подстановок для отдельно взятого вопроса. Как было сказано во введении, 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 описаан структура хранилища ответов. Можно говорить о том что 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, что бы 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, как правило, 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. Т.е. 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 процесс привязывать к вычислительному узлу. Однако этого не стоит делать при реализации остальных стратегий, так как коммуникационные затраты, вполне вероятно, перекроют полезное время вычислений, и тем самым лишь ухудшат результат.

Отметим что данная стратегия в общем случае конфликтует со стратегией отсроченного присваивания (поскольку СОП может нарушать независимость баз и вопросов).

Тестирование 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. представлены результаты тестирования.

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

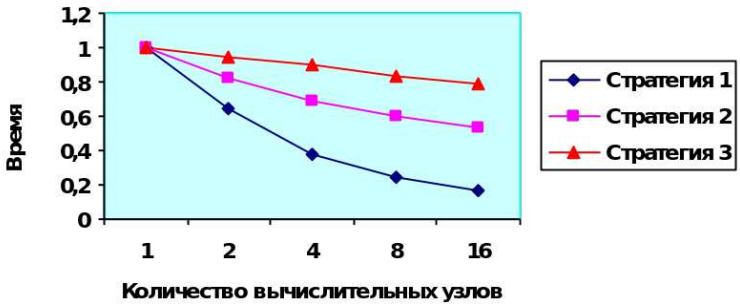

Рис. 2.5. Параллельные стратегии

2.2.11. Равенства

Для работы с равенствами как правило очень неэффективно напрямую использовать аксиомы равенства (рефлексивность, симметричность, транзитивность, подстановочность). Например, если формула содержит лишь один бинарный функциональный символ f и один бинарный атомарный символ A, то в языке ПОФ аксиомы равенства для такой формулы будут представлены следующим образом:

$$\begin{cases} \forall x\colon \boldsymbol{True} - \exists\colon x = x \\ \forall x_1, y_1, x_{2,y}\, 2\colon x_1 = y_1, x_2 = y_2 - \exists\colon f(x_1, y_1) = f(x_2, y_2) \\ \forall x_1, y_1, x_{2,y}\, 2\colon x_1 = y_1, x_2 = y_2, A(x_1, y_1) - \exists\colon A(x_2, y_2) \end{cases}$$

Таким образом, для каждого функционального и атомарного символа из формулы ставится в соответствие подформула-вопрос аксиома равенства. Явное использованеи таких аксиом во-первых, усложняет структуру формулы (лишние вопросы), во-вторых, увеличивает число шагов вывода, в-третьих, генерирует много (потенциально бесконечно) фактов в базе, возможно ненужных, мешающих выводу.

Отметим что данная проблема не так ужасна как в MP, поскольку в MP могут генерироваться лишние дизъюнкты, а это соответсвует генерированию лишних вопросов в ПОФ. В исчислении ПОФ же генерируются лишь атомы-факты, за которомы проще наблюдать, но тем не менее их много.

При классическом подходе, в соответствии с определением 3 подста-

новка θ является ответом на вопрос, тогда и только тогда, когда $A\theta \subseteq B$ где A - конъюнкт вопроса, а B - конъюнкт базы. Поиск ответов есть задача поглощения, для решения которой как правило используется алгоритм матчинга, об этом уже писалось выше. В случае ПОФ используется основной матчинг, а в случае НЭЭ используется полуосновной-матчинг.

Для решения проблемы основного матчинга без явного использования аксиом равенства имеется задача основного матчинга с равенствами, которая формулируется следующим образом [microsoft]: Для данного множества равенств $E(B)$, основного терма t и терма p, который может содержать переменные, необходимо найти множество подстановок θ, по модулю $E(B)$, такие что $E(B) \models t = p\theta$. Через $E(B)$ мы обозначем множество всех равенств в данной базе B. Две подстановки эквивалентны если их правые части попарно конгруэнты по модулю $E(B)$.

Для поиска таких ответов мы используем апарат теори систем переписывания термов [**Nipkow**].

2.2.12. Хранилище ответов

Хранилище ответов предназначено для эффективной организации работы с ответными подстановками и синхронизации с бэктрэкингом.

Каждому атому конъюнкта вопоса соответствует чанк возможных подстановок, матчащих этот атом с атомами из базы. Использование чанков связано с тем что необходимо точно определять на каком шаге какие подстановки были найдены. Для поиска ответа на вопрос, необходимо учитывать совокупность чанков соответствующих всем атомам конъюнкта вопроса.

На рисунке представлена схема хренения подстановок.

Далее из этой структуры выделяется ответ на вопрос. 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 несовме-

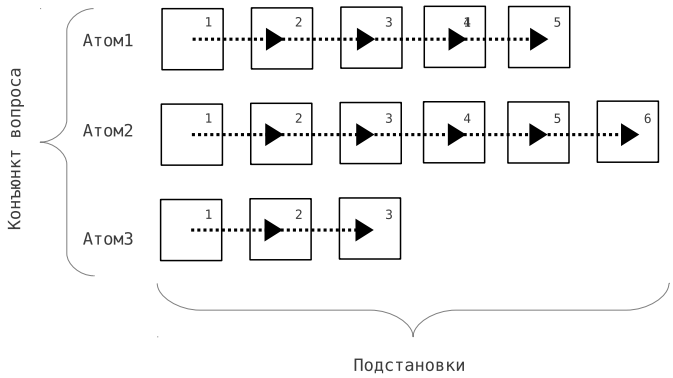

Рис. 2.6. Подстановки

стимы, 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.

2.2.13. Стандартная стратегия

Главное свойство стандртной стратегии заключается в её полноте. Для полноты вывода необходимо оргнаизовать полный последовательный пребор всевозможных ответов на все вопросы. Для этого используются возможности дерева состояний вывода и хранидища ответов.

Глава 3. Программная система

3.1. Реализация

Программная система является системой автоматического доказательства теорем в исчислении позитивно-образованных формул. Наша система относится ко второму классу систем АДТ, согласнно классификации приведенной во введении.

3.1.1. Архитектура системы

Картинка.

3.1.2. Основополагающие структуры данных и управление памятью

Подстановка. Подстановка есть список связей (binding) вида $X \to t$. В структуре Binding для X и t даны имена left и right соответственно. Определены следующие методы:

apply() — применить подставноку. В каждой связи терм left связывается с термом right, т.е. аргумент left ссылается на right.

reset() — сброс подстановки. Аргумент left являющийся переменной приводится в состояние NULL, а left являющийся НЭЭ не сбрасиывается. Применяется непосредственно после шага вывода, для того что бы освободить переменные для дальнейшего использования, но при этом оставить НЭЭ связанными.

fullReset() — сброс всей подстановки, включая НЭЭ. Применяется на этапе неудачной унификации, что бы вернуть все НЭЭ в прежнее состояни

3.1.3. Вспомогательные алгоритмы

Копирование подформул. В системе реализовано два тип копирования подформул. Первый тип предназначен для корректного копирования консеквентов и перемещения их к базе. Такое перемещение должно разорвать связь между переменными и связанными с ними термами, для того что бы освободить переменную для дальнешего использования в других шагах вывода, но при этом что бы в базу попали именно те термы с которыми данная переменная связана. Кроме того такой тип копирования нужен для параллельных стратегий, что бы переместить формулу в полностью независимый процесс. Данный тип копирования будем назвать жестким.

Второй тип копирования предназначен для правильной обработки НЭЭ с учетом бэктрэкингда. Поскольку при откатах необходимо востсанавливать информацию о том какой НЭЭ был связан с каким термом. Копирование НЭЭ проводится в прямую, вместе с привязанным к нему термом. Но потом такой НЭЭ в базе идентифицируется как связанный и работа ведется с привязанным термом. Такой тип копирования будем называть мягким.

Матчинг с НЭЭ. В сисеме реализован классический алгоритм матчинга адаптированный для случая содержания в термах НЭЭ. Для матчихщася термов строятся уравнения. Поведение НЭЭ определяется следующим образом. Если в уравнении НЭЭ находится в првой части, то его поведение совпадает с поведением универсальной переменной. Если слева, то НЭЭ не может быть доопределён до другой переменной. Если и справа и слева находятся НЭЭ, то возникает два варианта развития. Либо они совпадают и оба доопределяются до общего НЭЭ, либо несовпадают и алгоритм матчинга заканчивается неудачей.

Выбор вопроса.

Трансформация формул. Удаление фиктивынх кванторов. Расширение или углубление формулы.

Проверка доказательства. Имеется протокол вывода. Проверяется его корректность.

3.1.4. Системные предикаты и вычислимые термы

В логических языках программирования, например, Прологе [**Bratko**], 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. В данной работе 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.

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 языках логического программирования, например, Прологе, системные предикаты служат, в основном, для исполнения некоторых интерактивных действий: вывод информации на экран, чтение и запись файла, добавление и удаление фактов и др. Введем следующие системные предикаты: Next(L). Переход к вопросу, помеченному идентификатором L. Предикат помещается в коньюнкт корневой вершины консенквента 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 идентификатором 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. Предпо-

ложим, 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-ого элемента ряда Фибоначчи. На языке ПО-формул данная задача вычисления 10-ого элемента ряда формализуется следующим образом:

$$\forall\colon \boldsymbol{True} - \exists\colon f(1,1), f(2,2) - \begin{cases} \forall n, x, y\colon f(n,x), f(n+1,y) - \exists\colon f(n+2, x+y) \\ \forall x\colon f(10,x) - \exists\colon \boldsymbol{False} \end{cases} \tag{3..1}$$

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:

$$\forall n, x, y\colon f(n, x), f(n+1, y) - \exists\colon f(n+2, x+y)'(n+2), remFact(n) \quad (3..2)$$

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

$$\exists\colon f(1,1)'1, f(2,2)'2 \quad (3..3)$$

Удаление элементов из базы можно организовать без использования меток атомов, если использовать RemPatternFact:

$$\forall n, x, y\colon f(n, x), f(n+1, y) - \exists\colon f(n+2, x+y)'(n+2), remPatternFact(f(n, \_)) \quad (3..4)$$

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 интерпретиро-

ваться 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, или как в случае с равенствами не использовать апарат перписывания термов или кодированых деревьев. 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 связаны. В соответсвии со строковым представлением символа, ему задаётся его семантика в виде лямбда выражения.

3.2. Язык и трансляторы формул

3.2.1. Язык формул для прувера

БНФ?

3.2.2. Транслятор из формул ИП в ПО-формулы

Алгоритм трансляции.

3.2.3. Транслятор из TPTP в ПО-формулы

Для представления формул на языке предикатов первого порядка используется формат представления формул библиотеки TPTP.

3.3. Интерпретация полученных результатов

С каждым узлом ДСВ связывается некоторе событие (сообщение). В каждый момент времени по текущену состоянию ДСВ можно интерпретировать что же произошло, путём последовательного вывода сообщений начиная от корня и заканчивая листом, для каждой базы.

3.4. Интерактивные возможности

Например, при решении задачи о лифтах надо как-то добавлять информацию о поступающих выводах извне.

Как-то надо совместить это с Миниязычком.

3.5. Комментарии по предложенным стратегиям

Анализ методик показывает, что структура вывода — ДСВ тем более эффективна и тем выше её КПД чем больше элементов может содержаться в чанках. Размер чанка зависит от соответствующих консеквентов формулы. В терминах языка ПО-формул, более предпочтительными являются формулы имеющие как можно более крупные конъюнкты, и формулы имеющие глубину. Анализ задач из библиотеки TPTP показал что такой-то класс задач при формализации в языке ПОФ обладает описанным свойством.

При классическом подходе поскольку должны быть опровергнуты все базы, не стоит пытаться опровергать их одновременно, разростая тем самым дерево, можно сэкономить память строя однобокое дерево и доказывая всегда одну ветвь, таким подходом можно увеличить глубину пространства поиска.

3.6. Интерфейс и спецификация

Глава 4. Применение программной системы

В данной главе продемонстрированы примеры применения разработанных инструментальных средств для решения практических задач. В частности, рассматривается пример

4.1. Применимость

Как видно, каждый узел дерева состояний вывода олицетворяет шаг вывода. Кроме того, узел содержит очень много дополнительной информации, т.е. потребляет ресурсы памяти, причем с каждым шагом вывода потребление может увеличиваться. Таким образом с точки зрения потребляемости ресурсов прувер эффективнее использовать на задачах имеющих крупноблочную структуру, т.е., при формализации конъюнкты действительно должны быть конъюнктами а вырожденными одноэлементными или пустыми, древовидная структура действительно должна быть древовидной, глубокой, а не с глубиной 4. Таким образуом получится что каждый узел будет "обслуживать" довольно объемные и информативные куски данных. Одновременно с этим язык ПО-формул как раз и позиционируется как крупноблочный. Отсюда предположительно успех может возникнуть в задачах, формализуемых в языке ПО-формул таким образом, чтоб как можно сильнее выпячивалась крупноблочность. В самой крупной библиотеке формализованных задач из 10000 задач оказалось что 2000 обладают этим свойством. Поэтому тестирование и сравнение будет проводиться в следующих выборках: задачи которые предположительно наиболее удачно подходят для нашего прувера; и задачи которые предположительно являются неподходящими.

С другой стороны задачи с неограниченными переменными приводят к использования стратегии ленивой конкретизации, что в конечном итоге

может усложнить прозрачность логического вывода. Отсюда к перечисленным выше выборкам будут добавлены задачи с ограниченным и неограниченными переменными.

4.2. Тестирование

Тестирование системы проводилось на задачах из библиотеки TPTP (www.tptp.org). Данная библиотка является де-факто стандартом для тестирования систем АДТ. На момент написания работы библиотека содержала свыше 20000 задч, формализованных на языке предикатов первого порядка (FOF: first-order formula), конъюнктивной нормальной формы (CNF: conjunctive normal form), TFF.

Все задачи классифицированы по некоторым параметрам, а именно:

1. Рейтинг. Измеряется числом от 0.0 до 1.0. Задача с рейтингом 0 может быть решена любым прувером внесённым в библиотеку. Задача с рейтингом 1.0 ещё не была решена ниодной системой. Задачи с рейтингом выше 0.5 считаются весьма сложными. Тестирование имеет смысл проводить по всем видам рейтинга от 0.0 до 1.0.

2. Предметная область задачи. Например, теоремы математического анализа, алгебры, геометрии, головоломки.

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

4.3. Конкретные задачи

4.3.1. Задачи без неограниченных переменных

Из решенных задач время быстрее чем у Otter. Оттер выводит системную инфу всякую, поэтому думаю сопоставимо время на самом деле.

Задача; Рейтинг; Статус (Решил ли прувер); Комментарий.

MGT001; 0.12; Теорема (Да); 0.03с и 33 шага.

MGT008; 0.12; Теорема (Да); 0.007с и 6 шагов.

MGT010; 0.08; Теорема (Да); 0.007с и 13 шагов.

MGT015; 0.12; Теорема (Да); 0.1с и 7 шагов.

MGT007; 0.12; Теорема (Да); 0.01с и 25 шагов, есть ветвление.

GRA014; 0.00; Теорема (Да); 0.5с и 6797 шагов. У Otter уходит 7 секунд. При этом вывод системной инфы не такой огромный.

GRA024; 0.20; CounterSatisfiable (Неизвестно);

GRA023;

GRA022;

GEO169; 0.11; Теорема

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 логического вывода. Среди предложенных и реалиозованных стратегий имеются как адаптированные из других методов для ПО-формул, так и оригинальные.

Разработанный вид системы АДТ (пруверов) занимает промежуточное положение между интерактивными системами и классическими, позволя с одной строны производить достаточно автоматизированный логический вывод, с другой стороны вовлекать в процесс человека, тем самым используя основной дубль свойств человеко- и машинно-ориентированность.

Любые исследования в данной области являются актуальными. Формальное доказательство теоремы может иметь сколь угодно большую минимальную длину, поэтому любые методы позволяющие сократить число шагов вывода, а так же скорость их проведения важны. Многие современные и самые производительные системы АДТ уже столкнулись с проблемой сложности расширения и использования дополнительных знаний о задаче (универсализм систем). Данная работа является реализацией одного из способо решения этих проблем.

В рамках диссертации получены и на защиту выносятся следующие результаты:

1. Адаптированны некоторые общеупотребимые стратегии. Три типа параллельных стратегий, индексирование термов, один вариант разделения памяти (data sharing).

2. Реализоваы оригинальные стратегии предложенные именно для исчисления ПО-формул. Дерево состояний вывода, k,m-условие, стратегии

напарвленыные на решение проблем открытых переменных, варианты разделения памяти.

3. Реализован транслятор формул из языка библиотеки TPTP в язык ПО-формул.

4. Показана применимость

5. Пять

В диссертации показана возможность использования разработанных инструментальных средств при создании программных систем.

Несмотря на то, что в диссертации получен ряд положительных результато, необходимо сделать следующие замечания.

Основными особенностями, препятствующими внедрению разработанных инструментальных средств, на наш взгляд, являются:

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

2. Универсальная слабость прувера. То есть система показывает себя не достаточно сильно при решении задач без использования эвристик и модификаторов семантик.

Проблема 1 может быть частично или полностью решена путём развития пользовательских интерфейсов и путем просвящения пользователей. На проблему 2 стоит смотреть с точки зрения области применения систем. То есть не использовать систему там где необходимо решать универсальным методом, и использовать там где возможно использование сильных сторон исчисления ПО-формул.

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; и непосрдественно расширение класса решаемых

задач, в частности применение в области семантического веба (как достаточно хорошо развивающегося направления), биоинформатики.

Дальнейшая разработка системы связана с продолжнием повышения производительности поиска ЛВ.

Список литературы

[1] Васильев С.Н. Интеллектное управление динамическими системами. / Васильев С.Н., Жерлов А.К., Федунов Е.А., Федосов Б.Е. М.:Физматлит, 2000.

[2] Васильев С.Н. Об исчислениях типово-кванторных формул / Васильев С.Н., Жерлов А.К. // ДАН, Т. 343, N 5, 1995, с. 583-585.

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

[4] Thousands of Problems for Theorem Provers `http://tptp.org/`

[5] D Programming Language `http://www.digitalmars.com/d/`

[6] Alexandrescu A. The D Programming Language. / Addison-Wesley Professional; 1 edition, June 12, 2010

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

[8] Черкашин Е.А. Программная система КВАНТ/1 для автоматического доказательства теорем. канд. дисс. текст. / Черкашин Е.А. / ИДСТУ СО РАН, Иркутск, 1999. 155 С.

[9] Черкашин Е.А. Разделяемые структуры данных в системе автоматического доказательства теорем КВАНТ/3. / Черкашин Е.А. // Вычислительные технологии. 2008. Т. 13. С. 102-107.

[10] Butyrin S. A. An Expert System for Design of Spacecraft Altitude Control System. / Butyrin S. A., Makarov V. P., Mukumov R. R., Somov Ye.,

Vassilyev S. N. // Artificial Intelligence in Engineering. V. 11(1). 1997. P. 49–59.

[11] Graf P. Substitution Tree Indexing. / Graf P. // Proceedings of the 6th International Conference on Rewriting Techniques and Applications. 1995. P. 117-131.

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

[13] McCune W. Solution of the Robbins Problem / McCune W. // Journal of Automated Reasoning, V. 19(3), p. 263–276, December 1997.

[14] Robinson J. A. A Machine–Oriented Logic Based on the Resolution Principle. / Robinson J. A. // Journal of the ACM. (12). 1965. P. 23–41.

[15] Riazanov A. Implementing an Efficient Theorem Prover. / Riazanov A. / PhD thesis, The University of Manchester, 2003. 216 P.

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

[17] Wang H. Toward Mechanical Mathematics. / Wang H. // IBM J. Res. Devel. V. 4(1), 1960, P. 2–22.

[18] Wos L. Automated Reasoning: Introduction and Applications. / Wos L., Overbeek R., Lusk E., Boyle J. / McGraw–Hill, New York, 19

[19] Jean van Heijenoort / From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931, 1967 (1999).

[20] John Harrison / Handbook of Practical Logic and Automated Reasoning. Cambridge University Press; 1 edition (April 13, 2009)

[21] Peter Graf / Term Indexing (Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence). Springer; 1 edition (March 27, 1996)

[22] Geoff Sutcliffe / Progress in Automated Theorem Proving

[23] Roberto Cordeschi / The role of heuristics in automated theorem proving. J.A. Robinson's resolution principle. 1996

[24] W. Bibel, D. Korn, C. Kreitz, and S. Schmitt / Problem-Oriented Applications of Automated Theorem Proving.

[25] Reinhold Letz / P-SETHEO: Strategy Parallel Automated Theorem Proving

[26] AMD Verification. 2002

[27] Patric Blackburn / Automated Theorem Proving for Natural Language Processing

[28] Didier Bondyfalat / An Application of Automatic Theorem Proving in Computer vision. 1998

[29] Chandrabose Aravindan / Theorem Proving Techniques for View Deletion in Databases. 1999

[30] Adam Darvas / A Theorem Proving Approach to Analysis of Secure Information Flow

[31] Stefan Maus / Vx86: x86 Assembler Simulated in C Powered by Automated Theorem Proving. 2008

[32] Ulrich Kortenkamap / Using Automatic Theorem Proving to Improve the Usability of Geometry Software. 2004

[33] 2007

[34] Harison. Float. Minimum 2002

[35] Бутаков, Курганский.

[36] Чень, Ли. Математическая логика и автоматическое доказательство теорем.

[37] 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)

[38] Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1988)

[39] Bourbaki, N.: Theory of Sets. Paris, Hermann (1968)

[40] Moura, L., Bjorner, N.: Efficient E-matching for SMT Solvers. Proceedings of th 21st international conference on Automated Deduction: Automated Deduction, Bremen, Germany July 17–20, 2007) 167–182.

[41] Stikel, M.E.: Building theorem provers (2010)

[42] Schulz, S.: E — A Brainiac Theorem Prover.

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

[44] Smith D.E. A Source Book in Mathematics. 1984.

[45] Godel K. Über die Vollständigkeit des Logikkalküls. Doctoral dissertation. 1929.

[46] Гильберт Д., Аккерман В. Основы теоретической логики. Государственное издательство иностранной литературы. 1947.

[47] Логика и компьютер. Выпуск 5. 2004.

[48] Whitehead A.N., Russell B. Principia Mathematica. Cambridge University Press, Cambridge, 2nd edition (January 2, 1927).

[49] Newell A., Shaw J.C. Progrmaiing the Logic Theory Machine // In Proceedings of the 1957 Western Joint Computer Conference, IRE, 1957, P.230-240.

[50] 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, P.218-239.

[51] Wang Hao. Toward Mechanical Mathematics // IBM J. Res. Devel., Vol. 4, No 1, 1960, P. 2-22.

[52] Bourbaki, N.: Theory of Sets. Paris, Hermann (1968)

[53] Непейвода Н.Н. Прикладная логика.

[54] Братко И. Программирование для ИИ на языке Пролог.

[55] Database Systems: The Complete Book. 2000.

Приложение

Формальное описание миниязычка

Вывод

Таблица

Описание стратегии