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 совокупности с результатом Тьюринга и Черча это говорит о полуразрешимости теорий первого порядка. Идеи Эрбрана и по сей день лежат в основе многих методов.

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

Развитие аппарата математической логики и вычислительных машин естественно привело к созданию первых работающих на практике систем автоматического доказательства 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 положение 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 TPTP); 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 использует никакой содержательной информации 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 соответствии

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.

(Боюсь соврать) Из существующих систем АДТ для метода ПО-формул выделим ряд версий системы КВАНТ, разработанных раннее Е.А. Черкашиным, а также одну из последних версий разработанную Че и Ко (автором в частности), и Бутакова надо упомянуть. Предложенная в данной работе система является качественно новой, поскольку все предыдущие могут быть получены из неё путём спецификации. Система Бутакова конкретно предназначено для синтеза разборщика. Система Черкашина была разработано без учета функциональных символов, открытых переменных, обработки равенств, параллельных возможностей, разделения данных и др.

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 внутри 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 обусловлена 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 системы 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 направлении 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 лишь 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.

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

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

Теперь определим семантику ПОФ как семантику соответствующих формул языка предикатов первого порядка.

Определение 2 Пусть $A = \{A_1, \ldots, A_l\}$ есть конъюнкт и $\bar{x} = \{x_1, \ldots, x_n\}$ — множество переменных. Через $A^{\&}$ обозначим $A_1 \& \ldots \& A_l$, при этом $\boldsymbol{False}^{\&} =$

$False, \boldsymbol{True}^{\&} = True$ (пропозициональные константы). Через F^{FOF} обозначим образ соответствующей 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 $F^{\text{FOF}} = \forall x_1 \ldots \forall x_n \left(A^{\&} \to \left(G_1^{\text{FOF}} \vee \ldots \vee G_k^{\text{FOF}}\right)\right)$.

Любая ПОФ очевидно имеет структуру дерева. Таким образом, для удобства читаемости мы будем представлять их как древовидные структуры а также пользоваться соответствующей терминологии: 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 чётной (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

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

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 исчислении 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 универсум 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 некоторых переменных в каж-

дой 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 третьем шаге вывода первая база может быть опровергнута ответом $\{x \to y_1; y \to a\}$ на целевой вопрос Q_3. Опровергнутая база (подформула) для удобства представления может быть удалена из списка базовых подформул.

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

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

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

Опровержение закончено поскольку все базы опровергнуты.

Корректность и полнота 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 более человеко-ориентированным.

4. "Теоретические"кванторы $\forall x$ и $\exists x$ обычно не используются в формализации человеческих знаний, вместо этого чаще используются типовые кванторы $\forall x(A \to \sqcup)$ и $\exists x(A\&\sqcup)$ [**Bourbaki**], [**ICDS2000**].

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

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

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

8. Имеется естественный *ИЛИ-параллелизм*.

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

10. Благодаря 7, 9 доказательство может быть интерпретировано человеком. Такая интерпретируемость доказательства довольно важна с точки зрения человеко-машинных приложений. Таким образом, как говорилось 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 правила вывода $\boldsymbol{\omega}$. Такая модификация реализуется просто путём ограничений на применение правила вывода $\boldsymbol{\omega}$ и позволяет трансформировать классическую семантику исчисления ПО-формул в немонотонную, интуиционистскую, и т.п. Примеры использования таких семантик представлены в [**ICDS2000**].

1.3. Формальная информация о диссертации

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

Предметом исследования являются методы повышения производительности логического вывода, оригинально разработанные или адаптированные с других систем АДТ в новой системе 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-образованных 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-формул;
5. Исследование вопросов эффективного вывода с использованием предиката равенства;
6. Реализация инфраструктуры для участия человека в процессе логического вывода;
7. Апробация инструментальных средств в решении 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 системы 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). Разработана новый 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 исчислении 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 79 страницах машинописного текста, полный объем диссертации 80 страниц. В работе содержится 20 рисунков. Список литературы содержит 50 наименований.

Глава 2 посвящена стратегическому базису программной системы. Выделены некоторые проблемы ПО-исчисления, решение которых требовалось. Представлен ряд стратегий, адаптированных из существующих систем АДТ и новые стратегии для исчисления ПО-формул, рассмотрен вопрос решения задач с предикатом равенства.

Глава 3 посвящена аспектам реализации и использования системы. Системные предикаты, работа с арифметикой, трансляторы.

В главе 4 представлены методы и результаты тестирования системы. В частности дано описание библиотеки TPTP. Решена задача о лесах с применением внешних эвристик.

В заключении приводится список результатов выносимых на защиту.

Приложение содержит описание грамматики входного языка, пример протокола вывода для решения задач, сводную информацию о решенных задач из библиотеки TPTP.

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

В этой главе рассматривается задача адаптации существующих алгоритмов, используемых в различных системах АДТ для повышения производительности процесса поиска логических выводов, а также разработка других специализированных алгоритмов. Задача этой главы — реализовать полезные свойства ПО–исчислений при помощи известных продуктивных методик и алгоритмов поддержки различных этапов построения логических выводов. В качестве таких методик и алгоритмов выбраны: разделение данных (data sharing) [**Che2**, **Ryazanov2003**]; индексирование данных (term indexing) [**HARIndex**, **TermIndexingBook**, **pathindex**]; параллельные стратегии [**PSETHEO**]; стратегия ограниченных ресурсов [**Ryazanov2003** кэширование данных; расширена стратегия k–опровержения [**ICDS2000**, **dissChe**] до стратегии k, m–ограничения; для эффективной работы с предикатом равенства использованы результаты теории переписывания термов [**Nipkow**]. Предложен ряд стратегий работы с неограниченными переменными. Разработана структура данных — дерево состояний вывода, позволяющая реализовать одновременно некоторые из перечисленных методик.

В результате адаптации получены следующие результаты: значительное экономие памяти по сравнению с предыдущими версиями систем АДТ [**Cherkashin:diss:1999**], влекущее за собой экономию вычислительных ресурсов за счёт того, что предотвращается излишнее копирование термов и целых подформул; рост производительности с использованием параллельных стратегий (при условии применимости этих стратегий); сокращение пространства поиска за счет стратегий k, m–ограничения, сдерживающее разростание формулы из-за дизъюнктивного ветвления, и стратегии неограниченных переменных, позволяющих не перебирать эрбранов уни-

версум.

2.1. Анализ совместимости адаптируемых алгоритмов

Для того что бы алгоритмы адекватно использовали возможности ПО–формализма необходимо выявить его совместимые полезные свойства, а также свойства, не совместимые с адаптируемыми алгоритмами. Полезные свойства, которые обеспечены алгоритмически в рамках данной диссертации представлены во введении.

2.1.1. Свойства формализма ПО-формул, оказывающие влияние на адаптацию алгоритмов

Укажем на некоторые свойства ПО-формализма, сказывающиеся на производительности процесса поиска ЛВ и ... *Как оно сказывается?* в частичном разрешении которых он нуждается.

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

   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 аномалию в следствии некачественной или даже некорректной формализации задачи. Это, как правило, значит, что решение данной проблемы лежит вне прикладной области, оно необходимо для решения общематематических теорем, например, из библиотеки TPTP. Во-вторых, отказаться от использования функциональных символов (как это скромно сделано в [**ICDS2000**]), но наличие функциональных символов есть основа сложных задач. В-третьих,

в [**ICDS2000**] 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 указанных прикладных задачах нет нужды использовать СОП, т.к. в их формальном представлении нет открытых переменных. В общих же задачах сложнее найти какие-то особенности, которые применяются другими стратегиями, а значит конфликт частично разрешается просто разграничением областей применения системы *Или же алгоритма поиска подстановки для НЭЭ?*.

2. Язык ПО-формул в [**ICDS2000**] характеризуется как «достаточно однородной, но в то же время хорошо структурированный», а его исчисления «хорошо усваивают эвристики», т.е. базовая стратегия исчисления достаточно легко настраивается под конкретную задачу. Стоит заметить, что представление ПО-формул более разнообразно, чем, например, представление дизъюнктов, в силу использования разнородных сущностей в структуре формулы: база, вопросы, консеквенты вопросов. Разрешение данного вопроса *Какого? Я не понял. И в каком смысле "разрешение"* требует применения специальных методов доступа к данным неоднородным частям ПО–формулы.

3. Несмотря на то, что изначально представление формулы ИП в языке ПО–формул более компактно чем КНФ, применение правила вывода в ходе построения ЛВ при наличии дизъюнктивного ветвления, в общем случае, приводит к бóльшему усложнению структуры формулы, чем при применении правила вывода в методе резолюций. Таким образом, через некоторое количество шагов вывода размер ПО–формулы может оказаться в разы больше чем размер соответствующей КНФ. Но эта проблема практически полностью устранима технически, а именно описанными далее методами разделения общей оперативной памяти

данных, а также ограничивающей стратегией, которую пользователь выбирает сам.

2.1.2. Постановка задачи

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

2.2. Алгоритмы и структуры данных

2.2.1. Общие структуры данных

Это реализация или алгоритмы, причем тут типы? Алгоритм+Стр. данных = Программа (Вирт))

Тегирование структур данных В основе представления термов и атомов в системе лежит структура `Symbol`. Для дифференциации термов в этой структуре данных используются следующие теги: ATOM (атомарный символ), FUNCTION (функциональный символ), CONSTANT (константный символ), AVARIABLE (универсальная переменная), EVARIABLE (экзистенциальная переменная), UHE (Неопределённый Эрбрановский Эле-

мент), INTEGER (целочисленный символ), FLOAT (нецелочисленный символ), STRING (строковый символ). Данные значения теговых полей структур определяются перечислением:

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

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

1. Строковое представление символа. Необходимо для удобного (читаемого) вывода формулы на экран.
2. Тегирование символа, одним из перечисленных выше тегов.
3. Арность. Числовое значение указывающие на количество аргументов. Для констант, переменных, числовых символов и строк, значение равно нулю.

Символ идентифицируется его адресом в оперативной памяти ЭВМ.

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

Разные термы требуют соответствующих способов их обработки. Например, вместо универсальной переменной можно подставить другой терм, тем самым конкретизировав её, вывод на экран конкретизированной и неконкретизированной переменной отличается; обработка структура, представляющих функции, требует учёта наличия аргументов; вместо НЭЭ не может быть подставлена универсальная переменная и т.д. Поэтому способ работы с термом зависит от символа, лежащего в корне дерева представляющиего данный терм. Символ несёт необходимую информацию, в том числе тег *Повторение?*. Структура символа определяет в терме необходимое количество памяти под аргументы или возможные подстановки.

Терм представляется классической рекурсивной древовидной структурой, каждый узел которой содержит экземпляр структуры `GTerm` верхнего уровня *?*

и массив ссылок на его аргументы. Универсальная переменная (AVARIABLE) и неопределённый эрбрановский элемент (UHE) не содержат дочерних узлов, но используют единственный элемент одноэлементного массива ссылок-аргументов как ссылку на терм, который подставляется вместо термов AVARIABLE и UHE. При помощи этой ссылки реализуется конкретизация (подстановка???). Если эта ссылка указывает на NULL *Или НЕ является NULL??? см.* значит соответствующая переменная (или НЭЭ) конкретизированна некоторым термом. В противном случае переменная (или НЭЭ) является свободной для подстановки. Если переменная (или НЭЭ) конкретизированы, то терм представляющий переменную *М.б. уточнить, что за "представление переменной* в данный момент представляет конкретизирующий терм. При этом глубина конкретизации может быть сколько угодно большой, например при такой подстановке $\theta_d = \{x \to h_1, h_1 \to h_2, h_2 \to e\}$, где x — универсальная переменная, h_1, h_2 это НЭЭ, а e — константа. После применения θ_d, переменная x фактически конкретизирована до константы e, и дальнейшая работа с термом x производится как с константой e (в том числе вывод на экран). Для корректной работы с конкретизациям реализован метод `GTerm getValue()`, который возвращает самую глубокую конкретизацию данного терма. Если заданный терм t не является переменной/НЭЭ или, в противном случае, если переменная/НЭЭ некокретизирована, то значение `getValue()` совпадает со ссылкой на t. Иначе возвращается ссылка на самый глубокий терм, которым конкретизированна переменная/НЭЭ.

Описанный выше подход к конкретизации термов необходим для того, что бы корректно и эффективно производить откат подстановок, а для этого необходимо сохранять информацию о том, какая переменная/НЭЭ была конкретизирована, и каким значением/термом. Для того что бы распознать конкретизацию достаточно проверить первый аргумент массива аргументов на совпадение с NULL, а для того что бы откатить подстановку, значению аргумента просто присваивается NULL.

Подстановка. Подстановка (`Binding`) есть список связей вида $X \to t$. В структуре Binding для X и t заданы 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 сбрасывается. 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 состояние.

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

На рисунке 2.1. представлен пример структуры, образованной чанками.

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

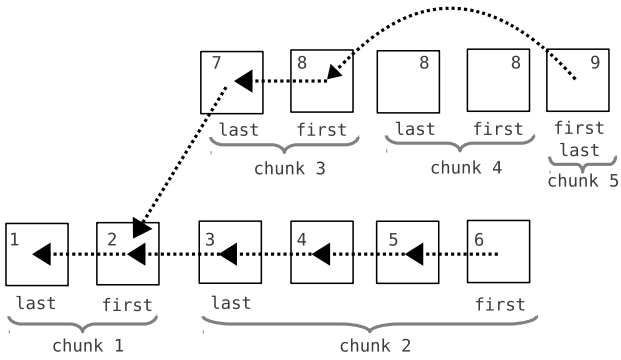

Рис. 2.1. Чанки

Поясним смысл данной структуры. В отдельный чанк заносится информация о состоянии текущего шага логического вывода, поскольку информация разнородна, то для каждого конкретного случая тип `T` конкретизируется. Связанные чанки образуют совокупность информации о шагах логического вывода.

2.2.2. Дерево состояний вывода

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

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

вую стратегию поиска ЛВ.

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

3. Реализовать стратегию разделения данных (data sharing) для случая расщепления базовых подформул.

4. Производить эффективное *В смысле программирования прувера или по скорости освоб* (и так и так) освобождение памяти.

Идея использования ДСВ базируется на анализе свойств классического подхода к *Чему?*. После каждого ответа на вопрос к первоначальной базовой подформуле добавляется пример консеквента этого вопроса: в базу добавляются соответствующие элементы узлов, непосредственно следующих за вопросом; к списку вопросов базы, в общем случае, добавляются новые вопросы; в случае дизъюнктивного ветвления база расщепляется. Таким образом, формула монотонно увеличивается , при этом сохраняя свою структур ДСВ будет использоваться для того, чтобы всегда иметь доступ к полной информации о текущем и прошлом состоянии формулы, а также для возможности возврата поиска вывода (backtracking) и подробного наблюдения *Какая-то* за процессом поиска ЛВ.

Более детально дерево состояний вывода есть такое дерево, которое обладает следующими свойствами: корень дерева есть одна из базовых подформул исходной ПО–формулы; все остальные узлы есть добавляемые консеквенты с примененным к ним подстановками–ответами и необходимым разыменованием переменных. Если приводить в пример определение правила вывода, то корень дерева — это база E, а узлы это $E2\theta$ *Лень формулу написать?∃* Таким образом если происходит расщепление базы то в соответствующем узле появляется ветвление. Теперь можно говорить, что каждая базовая подформула в формуле характеризуется соответствующим путём от листа ДСВ до её корня.

Как видно, каждый узел содержит достаточную информацию и для того, чтобы производить возврат поиска, для этого достаточно просто уда-

лять соответствующие узлы *А править состояния??*. Кроме того, такой подход реализует разделение данных (ссылок) на каждый консеквент, поскольку некоторые пути могут иметь общие подпути. Если какая-то база опровергнута, то можно удалить все узлы от соответствующего листа до ближайшей точки ветвления, поскольку оставшаяся часть пути всё ещё используется для представления другой базы. Количество листовых 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 *А одно корневое фиктивное ветвление нельзя?* ДСВ.

Для практических нужд *Этот абзац непонятен на 80%. Картинку в студию!* узел ДСВ содержит некоторую системную информацию:

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

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

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

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

5. Чанк использованных ответов.

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

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

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

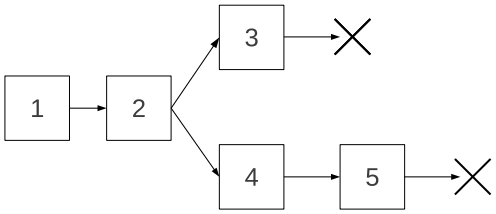

Рис. 2.2. Дерево состояний вывода для формулы из примера 2.

На следующем рисунке представлено ДСВ, как связанные чанки.

С точки зрения реализации ДСВ растёт от листов к корню *Не понял. Причем т*

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

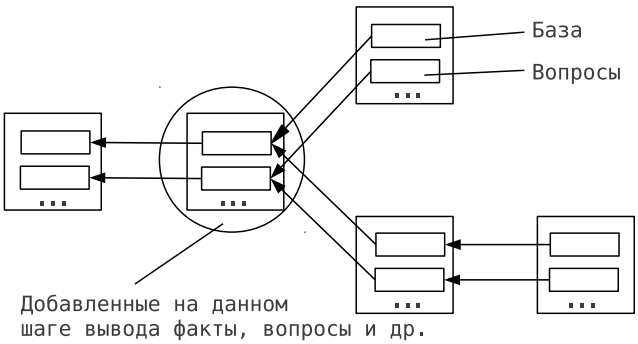

Рис. 2.3. ДСВ рис. 2.2., представленное в виде чанков

2.2.3. Супервизор

Супервизор наблюдает *???* глобально за всем ДСВ, обеспечивает алгоритмы информацией об ограничениях ресурсов, используемых эвристиках и т.п. В супервизоре находится список всех текущих листов ДСВ. Большинство стратегий реализовано на уровне супервизора, поскольку, в общем случае, необходимо использовать информацию о всей совокупности предшествующего или последующего вывода *Вот коряво то, что прошлое и будущее симмет*

Наличие такого субъекта управления *=СУпервизор, новая сущность?* обусловлено тем что некоторые частные события в процессе логического вывода могут повлиять на него *Субъект или управление?* в целом.

2.2.4. Разделение общей оперативной памяти структурами данных

Логический вывод, как правило связан, с получением новой дополнительной информации, ростом объема используемой оперативной памяти. Например, в метод Ррзолючии выводятся (синтезируются) новые дизъюнкты до тех пор пока не получится пустой дизъюнкт, а в методе доказательства ПО-формул производится насыщение баз фактами до тех пор, пока они не станут противоречивыми. Поскольку сложность формул может быть сколько угодно большой и даже минимальный вывод может иметь сколько угодно большую длину, имеет место проблема исчерпания имеющихся ресурсов вычислительной системы на хранение разрастающиеся структу-

ры формулы. Опыт показывает **??**, что автоматический вывод довольно быстро занимает всю имеющуюся в распоряжении оперативную память, и далее процесс вывода требует регулярное удаление излишков. За излишки можно принять любые части формулы. Например, в системах основаных на методе резолюций удаляются дизъюнкты, которые либо вообще не участвовали в выводе, либо не участвовали в нём определённое количество шагов, иногда удаётся определить что дизъюнкт больше не пригодится. В случае ПО–формул, излишками являются устаревшие факты, фиктивные вопросы. Таким образом, проблема экономии оперативной памяти является важной, особенно с учётом увеличения сложности задач *???*.

Для экономии памяти используются, во-первых проектирование компактных структур данных; во-вторых, методы разделения общих участков оперативной памяти (data sharing). В случае логических языков и конкретно языка ПО–формул использование методик хранения информации с разделением общей памяти является актуален. Исходя из некоторых общих особенностей представления языков первого порядка и представления ПО–формул, выделено и реализовано четыре методики разделения данных.

... *Будем говорить о каком-либо специальном менеджере оперативной памяти?* (БУДЕМ)

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 экономить большие объемы оперативной памяти 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 подхода намного меньшим потреблением процессорных ресурсов, но и меньшей эф-

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

Рассмтрим следующий фрагмент базовой ПО–формулы:

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

В данном случае вопрос $\forall x : A(f(e, x))$ имеет ответ $\theta = \{x \to g(t)\}$. К базе фактов добавляется $B(h(x), x)\theta$, т.е. $B(h(g(t)), g(t))$.

На рис. **??** представлен пример, демонстрирующий данную ситуацию с точки зрения мягкого разделения памяти.

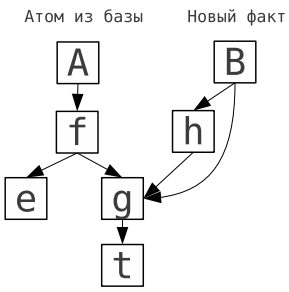

Рис. 2.4. ..

Разделение базовых подформул. ПО–формулы, в которых производится ответ на вопрос с дизъюнктивным ветвлением, расщепляются на несколько новых базовых подформул. Количество новых базовых подформул совпадает с количеством непосредственных дизъюнктивных подформул в консеквенте вопроса. В простом варианте реализации ЛВ [**dissChe**] такое расщепление требует копирования предыдущего состояния формулы несколько раз; такое копирование хотя и имеет линейную сложность [**Che2**], но всё равно естественно приводит к большим затратам памяти и процессорного времени, затрачиваемого для копирования. Разделение базовых подформул вполне реализуемо при помощи агрессивного разделения оперативной памяти термами. Однако, если формула предполагает достаточно сильное ветвление, сохраняется проблема наличия множества ссылок на разделяемые атомы баз, поскольку конъюнкт представляется как множество ссылок на атомы. Поскольку расщепление предполагает разделение общих частей баз, то имеет смысл разделять упомянутые выше ссылки. Данная стратегия реализуется за счёт средств ДСВ. Любая общая подветвь двух ветвей ДСВ является разделяемой. Рассмотрим небольшой пример. Пусть имеется следующая базовая подфомрула:

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

Обозначим первый и второй вопросы через Q_1 и Q_2 соответственно. ДСВ для вывода данной формулы представлено на рис. **??**.

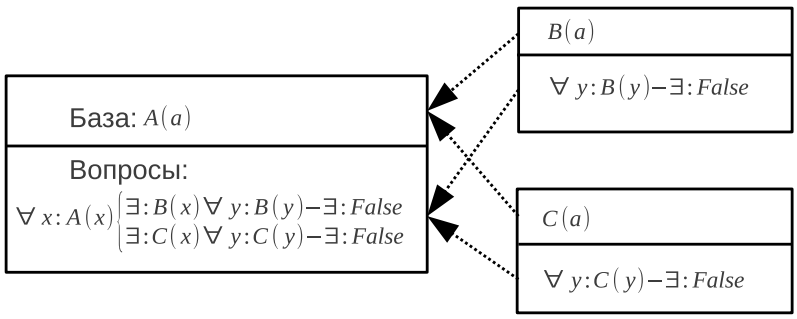

Рис. 2.5. ..

Разделение переменных и неопределенных эрбрановских элементов. Данная методика предназначена для высокопроизводительного применения подстановки ко всей подформуле, т.к. все одноименные переменные в формуле представляют собой один участок в оперативной памяти. О неопределенных эрбрановских элементах сказано ниже *А надо выше, после GTerm*. 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: достаточно одной операции присвоения над одной переменной, чтобы установить замену всех таких переменных в поддереве.

Удаление неиспользуемых фактов. Ниже, показано что стандартная стратегия поиска ответных подстановок основано на том что каждого атома из конъюнктов вопросов производится попытка мэтчинга его с каждым атомов-фактом из базы. Поскольку, количество вопросов и длина конъюнктов конечны, то нетрудно определить для атома-факта из базы, мэтчится ли он хотя бы с одним атомом из вопросов. Если нет, то такой атом-факт можно удалить из базы, поскольку он вообще не используется в поиске ответных подстановок. Удаление ненужных фактов так же позволяет экономить память.

Веса подформул. Под весом терма или подформулы понимается количество узлов в дереве, представляющем терм или подформулу. Анализ веса позволяет сдерживать разрастание формулы, и, соответственно обеспечить дополнительную экономию потребляемой памяти. Для этого из возможных ответов на вопрос приоритет отдаётся тому, который приводит к формуле наименьшего веса.

... *Что насчет управления комбинациями методик? Откат и вес? Как задавать эти ко*

2.2.5. Стратегия ограниченных ресурсов

Стратегия ограниченных ресурсов реализована в большинстве современных систем АДТ, в том числе и в Вампире []. В данной системе подобная стратегия напрямую ??? зависит от текущего ДСВ и описывается следующим образом. Ветка ДСВ растет до тех пор, пока не наступит ограничение объема оперативной памяти, разрешенной для использования, либо пока не исчерпается определенное время, либо пока не будет произведено определенное количество шагов вывода. Если наступил предел использования ресурсов, то осуществляется откат назад и выбор других ответов, т.е. используются другие варианты построения дерева.

2.2.6. Индексирование данных

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

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

Отсюда необходимо разработать методы, позволяющие в формуле производить эффективный поиск необходимых её (термов, вопросов, овтетов).

Для решения данной проблемы обратимся к существующему опыту.

Индексирование термов Основной структурой, используемой для представления формул является обобщенный терм. Он используется для представления элементов конъюнктов, кванторных переменных, обеих частей подстановок. Поэтому актуальной задачей является эффективный поиск обобщенных термов в формуле по заданным критериям.

В информационных технологиях поиск данных часто разрешаются, например, с помощью методов индексирования данных, применяемых широко в реляционных базах данных [**Ulman**].

В нашем случае основной объект индексирования — это обобщенный терм, который является древовидной структурой, индексирование которой методами, используемыми в реляционных БД, неэффективно [**Graf**]. Поэтому используются нижеописанные подходы.

Индексирование термов к настоящему времени хорошо исследовано, как в рамках определенных систем АДТ, так и абстрактно. В частности по данной теме существует ряд интересных работ, в том числе [дискриминационное дерево и пути, подстановочное дерево, и книжку Графа про индексирование]. Представленные в этих работах 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 *Обновить надо?*, но дополнительно необходимы методы индексной поддержки НЭЭ-унификации: критерий наличия термов, включающих заданный подтерм; различные количественные критерии: вес, глубина, арность. С учетом перечисленных требований а также того факта, что как правило, в базе находятся основные термы *Через абзац будет использоваться "основной пример" кругом.*, качестве основы методики индексирования выбрано индексирование путями []. Кратко опишем её суть, как она описана в [].

Для каждого символа входящего в терм составляется список так называемых *путей*. Путь — это последовательность чередующихся симво-

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

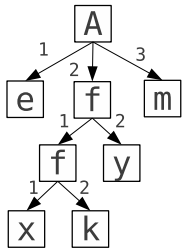

Рис. 2.6. ..

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 $A(a, f(x, b))$.

Методы индексирования термов в настоящее время широко 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. Поскольку количество вопросов не так велико как количество термов, то достаточно использовать возмодности обычного словаря.

Индексирование других частей формулы Теперь опишем другие проблемы, решаемые при помощи индексирования. Предположим что пользователем задана 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). Для повышения производительности поиска ЛВ необходимо упростить в дальнейшем доступ к таким вопросам, чтобы каждый раз повторно не проверять все вопросы на наличие определенного свойства. Для этого достаточно сделать словарь (map) в котором каждому описанию вопроса *Что за описание? как оно со* соответствовал бы указатель на данный вопрос. В принципе, в стратегиях задавать

Индексирование ПО–формул Доказательство крупных формул может производиться в несколько этапов, с сохранением промежуточных результатов на жесткие носители информации. Сам вылогический вывод формально является цепочкой формул. Для анализа описанных ситуаций необходимо проверять формулы на наличие некоторых свойств. Поскольку, ПО–формула как и обобщенный терм имеет древовидную структуру, то для её индексирования применимы описанные выше методы индексирования путями. Однако в данном случае индекирование путями требует доработки, поскольку, в случае термов путь однозначно задавался символами и номерами дочерниих узлов, то в случае ПО–формул номера дочерних узлов сохраняются, а вот символу может соответствовать любая формально описуемая характеристика узла ПО–формулы (типового квантора). [Буду дописывать данный параграф. Есть поле для размышлений. Можно на статью небольшую сделать. В книгах про индексирование подобные задачи описаны как нужные.]

2.2.7. Неограниченные переменные

Проблема неограниченных переменных описана выше. Кратко опишем её суть. Согласно определению 3правило вывода ω применимо в том случае

если вопрос $\forall \bar{y}\colon A$ к базе $\exists \bar{x}\colon B$ имеет *ответ* θ такой что θ есть подстановка $\bar{y} \to H^\infty$ и $A\theta \subseteq B$. В случае если все перменные из $\bar{y}$ содержатся в конъюнкте A, то поиск соответствующих элементов из H^∞ производится с помощью алгоритма поглощения. Если же среди $\bar{y}$ имеются переменные, которые не входят в A, то такие переменные называются *неограниченными*, и не используются в алгоритме поглощения. Но подстановка для них необходима, причём любой элемент эрбранова универсума является формально корректной подстановкой. Проблема в том, что в общем случае (при наличии функциональных символов) эрбранов универсум является бесконечным множеством, и какой именно элемент из него необходимо выбрать неизвестно.

Решение данной проблемы не является тривиальной. Нами предложенно несколько подходов.

Стратегия ленивых конкретизаций

Суть ленивой конкретизации заключается в следующем. В качестве подстановки для неограниченной переменной выбирается не конкретный элемент эрбранова универсума, а неопределенный эрбрановский элемент (НЭЭ), который, в дальнейшем, исходя из нужд вывода, постепенно конкретизируется до основного терма, либо в некоторых ситуациях так и остается недоопределенным. Поясним некоторые слова из предыдущего предложения. В данном случае «нужды вывода» это возможность ответа на вопрос, и НЭЭ конкретизируется таким образом, что бы появился новый ответ на вопрос, т.е. решается следующая задача: до какого терма конкретизировать НЭЭ, что бы появился ответ на вопрос?. Отметим, что НЭЭ первоначально появляется как часть ответа в одном вопросе, а его конкретизация производится при поиске ответов на другие вопросы. Под «постепенной конкретизацией» понимается процедура ленивых вычислений: НЭЭ доопределяется на столько точно, на сколько этого достаточно для ответа на текущий вопрос, т.е. конкретизация может быть неполной (т.е. не конкретизирующая НЭЭ до основного терма), например НЭЭ h конкретизируется до $f(h_1)$, где h_1 новый НЭЭ.

По своей природе НЭЭ схож со [свободной] переменной в том смысле, что он изменяем *Уточни термин.*, однако все такие изменения должны быть направлены только на конкретизирование НЭЭ, т.е. НЭЭ заменяется только на некий терм (возможно тоже содержащий НЭЭ), либо на другой НЭЭ, но не на переменную. Однако, при замене переменной на НЭЭ, НЭЭ обладает всеми свойствами терма.

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

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

На первом шаге вывода получен 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 получен 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 третьем шаге получен тривиальный ответ на третий вопрос, и h_2 доопределяется до a.

Однако существуют особые ситуации. Рассмотрим следующий пример.

Пример 3 ()

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

Данная формула имеет вывод, например такой. Получаем ответ на первый вопрос подстановкой $\{x \to e, y \to e\}$, в результате в базу попадают факты $S(e), M(f(e)), T(e)$; ответ на второй вопрос есть $\{x \to e\}$, и в базу попадает $Q(e)$; полученных фактов в базе недостаточно для ответа на целевой вопрос, поэтому вновь отвечаем на первый вопрос, при этом возможно

несколько вариантов ответа, но, что важно, переменную y необходимо заменить на $f(e)$. Выберем, например, следующий ответ $\{x \to e, y \to f(e)\}$ и в базу попадет факт $S(f(e))$ после чего на целевой вопрос ответ получается. Заметим, что на первый вопрос необходимо обязательно выбирать те подстановки, которые содержат $y \to e$ и $y \to f(e)$, они необходимы для ответа на второй и третий вопросы, соответственно. Формула устроена таким образом что первый и второй вопросы всегда имеют новые ответы.

Теперь рассмотрим работу стратегии ленивой конкретизации. Ответ на первый вопрос будет следующего вида $\{x \to e, y \to h_1\}$, где h_1 — НЭЭ, и в базу попадают следующие факты $S(h_1), M(f(e)), T(e)$. Ответ на второй вопрос — $\{x \to e, h_1 \to e\}$, в котором h_1 доопределяется до e, и, соответственно, находящийся в базе факт $S(h_1)$ доопределяется до $S(e)$. С этого момента начинается выполнение, фактически, циклической операции, поскольку целевой вопрос не имеет ответа: вновь получаем ответ на первый вопрос, и вновь в базу попадает факт $S(h_1)$, который при ответе на второй вопрос вновь 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 существуют ситуации, когда с формальной точки зрения НЭЭ конкретизируется корректно, но не там где это требуется, например раньше, чем это необходимо при ответе на другой вопрос. Либо НЭЭ может конкретизироваться там где это уже не требуется. Из подобных примеров видно, что ленивая конкретизация не может использоваться на прямую. Необходимы дополнительные средства обеспечения логического вывода, учитывающие описанные только что проблемы.

Ограничение количества конкретизаций Два НЭЭ будет называть *подобными*, если они получены в результате выбора подстановки для одной и той же переменной. Такая ситуация имеет место быть если на один и тот же вопрос с неограниченными переменными произведено несколько ответов, для неогарниченных переменных все разы в качестве подстановки

выступает новый НЭЭ.

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

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

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

Содержательно такой подход означает следующее. Можно считать что

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

В данном случае можно заметить сходство со стратегией ленивых конкретизаций, без каких-либо ограничений. Преимущество данного подхода заключается в том, что выражение, содержащее НЭЭ всегда сохраняется, т.е. не может быть утрачено за счёт «неугодных подстановок».

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

В случае с дизъюнктивынм ветвлением, вся совокупность оветов на вопрос означает не просто появление бесконечного конъюнкта, а бесконечно множества конъюнктов, поскольку база ещё и расщепляется. Т.е. конкретизация НЭЭ приводит не просто к порождению нового атома в конъюнкте, а к очередному расщеплению формулы, в той точке логического вывода, в которой впервые появился доопределяемый НЭЭ. Данную точку легко отследить благодаря структуре ДСВ, в которой сохраняются все события произошедшие на каждом шаге вывода. Однако, ветвление целесообразно проводить не в этой точке, а во всех листах дерева, корнящегося из этой тчоки. Поскольку вставка втевления в середину вывода потребует копирования для каждого нового узла той части вывода которая была уже прозведенаа. На рис. 2.7. схематично показано некоторое ДСВ.

После конкретизации НЭЭ, получаем структуру ДСВ, изображенное

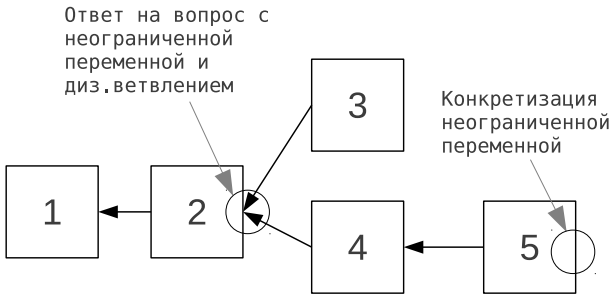

Рис. 2.7. ..

на рис. 2.8.

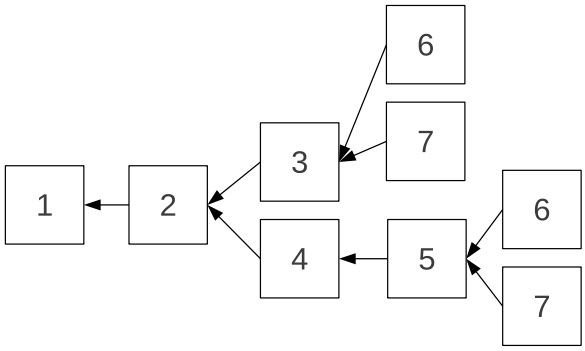

Рис. 2.8. ..

В случае подформулы-вопроса, содержащего НЭЭ, конкретизация НЭЭ требует порождения новой подформулы-вопроса, отличающейся от исходной только в точке содержания НЭЭ. Возможно появление множества вопросов, усложняющее в целом структуру формулы, и её вывод. Под усложнением структуры в данном случае понимается именно наращивание количества вопросов. Абсолютный размер формулы (в байтах) меняется незначительно, поскольку используются стратегии экономии памяти. Новую подформулу-вопрос целесообразно вставлять в той точке вывода (перед этим узлом), где первоначально добавился вопрос с НЭЭ. Предположим что на рис. 2.7. вопрос с НЭЭ впервые появляется в узле 2. Тогда если конкретизация НЭЭ потребуется в любом из последующих узлов, дерево преобразуется в вид, как на рис.2.9.. При этом новый узел 6 содержит только новый порождённый вопрос.

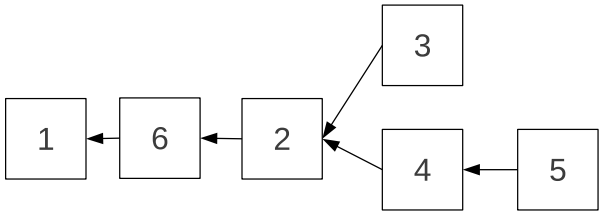

Рис. 2.9. ..

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

Но что делать, если дополнительные знания о задаче не позволяют сформулирова

Отметим, что стратегии, основанные на использовании НЭЭ, стоят особняком для многих других стратегий. Во-первых, при использовании этой стратегии нарушается независимость базовых подформул, поскольку один и тот же НЭЭ может быть в разных подформулах: доопределение общего НЭЭ в одной подформуле автоматически приводит к доопределению его в другой подформуле. Этот факт, в частности, делает зависимыми процессы поиска ЛВ в базовых подформулах на параллельных вычислительных архитектурах. Он также противоречит некоторым эвристическим уточнениям алгоритмов и стратегий, например, ...Пошел простой пример стратегии..

Стратегия фильтрации эрбрановского универсума

Другим вариантом (помимо стратегии ленивых конкретизаций) решения проблемы неограниченных переменных, является стратегия фильтра-

ции эрбрановского универсума. Данная стратегия позволяет избавиться от изложенных в начале главы проблем, но несколько расширяет пространство возможных ответов. Суть данной стратегии заключается в следующем. В консеквентах, атомы, которые 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 условию применения правила вывода (**??**), поскольку никода не выполнится процедура матчинга при поиске ответов. Будем использовать данное свойство в стратегии. В качестве ответа для неограниченных переменных используются такие термы, что атом полученный подстановкой переменных на эти термы будет основным примером найденной унификации. Такой подход позволяет заранее сузить пространство эрбрановского универсума, фильтруя заведомо бесполезные ответы.

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

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

и

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

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

Такой подход имеет сходство со стратегий ленивых конкретизаций, в которой конкретизация НЭЭ по сути означает сужение эрбранова универса, до множества основных примеров конкретизации. В данном же случае ограничения на ЭУ накладываются сразу, исходя из потенциальных во-

просов. В случае стратегии ЛК, конкретизация как раз производится при поиске ответов на эти потенциальные вопросы.

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

2.2.8. Стратегия k, m–ограничения

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

k, m–опровержение. Заданный ответ выбирается в случае, если за последующие k шагов ЛВ будет опровергнуто как минимум m баз. Подобная стратегия, а именно k–опровержение, первоначально предложена в [**ICDS2000**] и реализована в системе КВАНТ/1 Черкашина Е.А. [**dissChe**], где показана её состоятельность. В данной системе эта стратегия расширена вторым параметром m. Она позволяет сдерживать разрастание пространства поиска вывода, т.е., сдерживает излишнее ветвление ДСВ, что в некоторых случаях приводит к многократному усложнению вывода.

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

пенно увеличивается, а m уменьшается.

k, m–конкретизация. Иная спецификация стратегии k, m–ограничения, формулируется так. Ответ на вопрос принимается, если за последующие k шагов будет доопределено m НЭЭ. Эта стратегия также направлена на то чтоб ограничить сложность представляемой формулы. С недоопределенным НЭЭ ассоциируется много дополнительной информации и условий, описанных в главе о неограниченных переменных, что на уровне реализации влияет негативно. Поэтому чем больше и быстрее НЭЭ будут доопределены, тем лучше.

Особенности реализации стратегии k, m–ограничения Основной для реализации данной стратегии являются описанные выше ДСВ и супервизор. Параметры k, m и собственно условие задаются в супервизоре и привязывается к определённом узлу ДСВ. Супервизор проверяет все условия на каждом шаге ЛВ. И в положительном случае (когда условие выполнено) производится возврат поиска (backtracking) назад, т.е. производится последовательное удаление узлов дерева в обратном порядке (от листа в направлении корня). Перед удалением каждого узла производится разконкретизация НЭЭ, полученных в этих узлах и возврат использованных вопросов в стадию активных (т.е. возможных для применения).

2.2.9. Кэширование результатов

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

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

Это позволяет делать корректные возвраты поиска с учетом сохранения информации о примененных ответах.

Тут можно было б таки описать, как кэш используется. Что строиться - понятно.

2.2.10. Параллельные стратегии

Повышать производительность можно при помощи вышеописанных интенсивных методов, ориентированных на оптимизацию использования вычислительных ресурсов, а также при помощи экстенсивных методов, базирующихся на вовлечении в процесс дополнительных вычислительных ресурсов. В частности, это становится актуальным ввиду широкого распространения многоядерных вычислительных систем общего назначения, в частности, рабочих станций. Одним из популярных экстенсивных методов повышения производительности является разработка версий программ АДТ для кластерных архитектур в параллельном режиме исполнения ветвей подпрограмм. Рассмотрим методики и стратегии построения параллельных реализаций алгоритмов поиска ЛВ в исчислениях ПО–формул.

Предложены следующие стратегии для построения параллельных схем алгоритмов поиска логического вывода.

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

в отдельном вычислительном процессе или на отдельном вычислительном устройстве.

Таким образом, первая стратегия, реализуемая в виде параллельной схемы алгоритмов, формулируется следующим образом: каждая базовая подформула , содержащая только основные термы в базе, опровергается независимо от других базовых подформул, а значит для этот процесс выделяется в отдельный параллельный независимый от других процесс, синхронизируемый с другими процессами только на этапах его создания в момент расщепления формулы и завершения в момент установления выводимости/невыводимости. Во время жизни этого процесса к нему может поступать асинхронный сигнал завершения от супервизора, обозначающий, что выполнение процесса далее не имеет смысла, например, формула является неопровержимой, что было доказано в какой-либо другой ветка поиска доказательства, или пользователь остановил выполнение программы. Для программной реализации алгоритмов данной стратегии созданы подпрограммы жесткого копирования и маршаллинга/демаршаллинга *Представления динам* (следующая глава), что бы полностью скопировать базовую подформулу и обрабатывать её в отдельном процессе независимо, т.е. не разделяя оперативную память.

... *Думаю этот раздел немного покажет тебе на сколько подробно можно и надо писать*

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.

... *М.б. тут тоже рассказать в общих чертах о варианте реализации, например, что это не*

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 описана 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 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). [...] *А это не надо было где-то раньше сказать. Например в разд. "*

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. 2.10. представлены результаты данного тестирования.

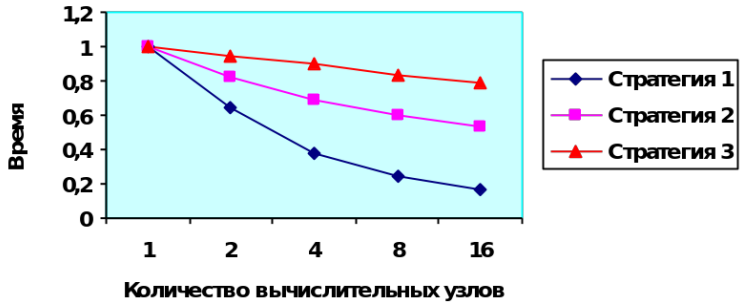

Рис. 2.10. Результаты тестирования параллельной реализации алгоритмов АДТ, согласно представленной стратегии

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

2.2.11. Равенства

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

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

Как видно, для каждого функционального и атомарного символа из формулы ставится в соответствие подформула-вопрос — аксиома равенства. Явное использование таких аксиом, во–первых, усложняет структуру формулы (появляются лишние вопросы), во-вторых, значительно увеличивает число шагов вывода, в-третьих, генерирует много (потенциально бесконечно) фактов в базе, возможно не участвующих в выводе, а также мешающих выводу. Отметим, что данная проблема не так ужасна как в MP, поскольку в MP, в общем случае, порождаются дополнительные дизъюнкты, а это соответствует порождению дополнительных вопросов в ПОФ. В исчислении ПО–формул порождаются лишь атомы-факты, за которыми проще наблюдать *Вот надо подобрать четкий эквивалент.*, но тем не менее их много.

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

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

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

С помощью правил переписывания генерируются новые атомы-факты, эквивалентные, уже имеющимся в базе. В сочетании со стратегией удаления неиспользуемых фактов (стратегия экономии памяти), очевидно, что база не будет захламляться ненужными сгенерированными фактами.

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

2.2.12. Хранилище подстановок

Хранилище подстановок предназначено для эффективной организации доступа алгоритмов к ответными подстановками и возможности организации механихзма бэктрэкинга, для приведения возможных ответов в предыдущие состояния (состояния на предыдущих шагах вывода).

Каждому атому конъюнкта вопроса соответствует чанк возможных подстановок, обеспечивающих матчинг этого атома 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 использовать алгоритм композиции подстановок для каждого атома из вопроса.

На рис. 2.11. представлена схема хранения подстановок для каждого атома.

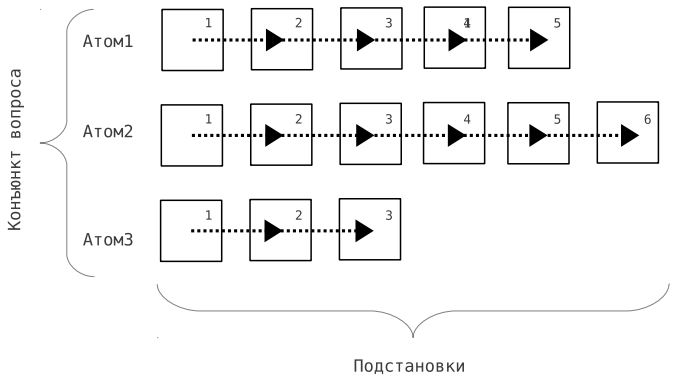

Рис. 2.11. Подстановки

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

Перебор подстановок из чанков производится последовательно *Как декартово пр* это позволяет сохранить полноту *Можно ли за счет состояния порождать др. состояние*

2.2.13. Стандартная стратегия поиска логического вывода

Главное свойство стандартной стратегии поиска логического вывода заключается в её полноте. Для полноты вывода необходимо организовать полный последовательный перебор всевозможных ответов на все вопросы.

Для этого используются возможности дерева состояний вывода и хранилища ответов. ... *Надо, я так понял, стратегию описать здесь подробно. Или зачем тогда разд*

Глава 3. Программная система

Глава 3. посвящена вопросам реализации системы автоматического доказательства первопорядковых теорем в ислислениях позитивно–образованных формул. В качестве исходной информации для реализации выступают структуры данных и алгоритмы, описанные в предыдущей главе диссертации. В результате реализации создана программная система АДТ, ориентированная как на решение задач, возникающих при решении практических приложений, так на задачи, традиционно, считающиеся математическими (абстрактные).

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. Вспомогательные алгоритмы

Почему сразу со вспомогательных алгоритмах начал изложение?

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*.

Копирование подформул в процессе шага вывода, требует во-первых разыменования всех экзистенциальных переменных, и всех неконкретизированных универсальных переменных. Поскольку структура копируемых подформул является древовидной, то для копирования применяется рекурсивный алгоритм, начинающий свою работу с корня дерева. Опишем подробно алгоритм.

В начале работы алгоритма задан пустой словарь переменных, которые необходимо разыменовать и соответствующих им разыменованных переменных. Для текущего узла (типового квантора), для его кванторных переменных создаётся новый список переменных, той же длины что и исходный, и того же типа (EVARIABLE или AVARIABLE), полученные соответствия заносятся в словарь. Далее при копировании конъюнкта, если встречается неконкретизированная переменнаяи она уже входит в словарь, то её копией будет соответствующее ей в словаре значение, это делается для того что бы одинаково разыменовать одинаковые переменные. Если переменная конкретизирована, то копируется значение функции `getValue()`. Отметим, что не может быть ситуации, когда встречается неконкретизированная переменная, которой нет в словаре, поскольку используются только замкнтые формулы, а в словарь последовательно добавляются все переменные управляемые кванторами. Процедуре копирования дочерних узлов

передаётся текущий словарь.

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 правой 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.

Дано два терма q и b, соответственно из базы и конъюнкта вопроса. Отметим, что поскольку b это терм из базы, то он не содержит универсальных переменных. Задан пустой ответ a.

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

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

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

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

5. Если q тегирован как FUNCTION. Если b тоже тегирован как FUNCTION, то если корневые символы q и b не совпадают, то вернуть NULL. Если совпадают, то это одноименные функции, одинаковой арности. Последовательно провести алгоритм матчинга для всех попарно соответствующих термов. Если алгоритм возвращает не NULL, то результат применить и добавить к a. Если получен NULL, то сбросить a и вернуть NULL. Если b — это НЭЭ, то применить подстановку $\{b \to q_s(h_1, ..., h_n)\}$ и применить алгоритм ко всем парам (h_i, q_i), где q_s это корневой символ s, n — арность q, h_i — новые НЭЭ, $i = \overline{1,n}$. Полученные результаты применить и добавитьк a, вернуть a.

Композиция подстановок.

Выбор вопроса.

Обработка формул. Удаление фиктивных кванторов, и т.д.

Проверка доказательства. Имеется протокол вывода. Проверяется его корректность.

3.1.3. 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 разделе рассматривается использование системных предикатов для управления логическим выводом в процессе построения автоматического доказательства теорем в исчислении ПО-формул, то есть как способ задания дополнительных знаний о задаче в виде модификаторов стратегии, используемой по умолчанию.

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

Очищение формулы. Другим типом задач являются задачи с некоторой дискретизацией шагов. Например, если моделируется переход системы из одного состояния в другое во времени, шаг такого перехода может быть эквивалентен нескольким шагам правила вывода . Часть выведенной информации, отвечающая за сам процесс перехода, может быть удалена, а другая часть, отвечающая за описание нового состояния, сохраняется. Примером может служить задача планирования движения робота в дискретном пространстве. Часть информации устаревает, и ее необходимо ре-

гулярно 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:

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

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

$$\exists\colon f(1,1)'1, f(2,2)'2 \tag{3..3}$$

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

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

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 интерпретироваться пользователем или возможно иной программой управления ЛВ, как некоторые входные данные для принятия решения. Среди выводимой информации отметим следующие: текущий шаг вывода, имя базы, имя вопроса, ответная подстановка, количество фактов в базе, количество опровергнутых баз, количество вопросов, количество потенциальных ответов, время ЛВ и печать некоторых термов.

Вычислимые термы. При решении прикладных задач может быть что заранее известна семантика формулы, область интерпретации и т.д. Отсюда некоторые сложные термы можно просто вычислять, а не выводить классическим образом, или как в случае с равенствами не использовать апарат переписывания 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 явля-

ются 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.

ALG211+1; 0.12; Теорема (Да); 0.05 и 40 шагов.

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); 1.2с и 5201 шагов. У Otter уходит 7 секунд. При этом вывод системной инфы не такой огромный.

GEO175+1; 0.12; Теорема (Да); 0.01 и 19 шагов.

GEO175+2; 0.12; Теорема (Да);

GEO177+1; 0.25; Теорема (Да); 0.01 и 41 шаг.

GEO178+1; 0.04; Теорема (Да); 0.01 и 42 шага.

GEO179+1; 0.08; как обычно

GEO180+1; 0.21;

GEO180+2; 0.17;

GEO183+1; 0.08; ; 28 шагов.

GEO184+1; 0.08; ; 9 шагов.

GEO186+1; 0.12;

GEO187+1; 0.21;

GEO188+1; 0.17;

GEO189+1; 0.25;

GEO190+1; 0.21;

GEO195+1; 0.29;

194+1, 193+1, 197+1, 198+1, 200+1, 201+1, 202+1, 203+1, 204-216, 236,

SET913+1; 0.04; ..

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.

[56] Ian Wehrman, Completion without failure.

Приложение

Формальное описание миниязычка

Вывод

Таблица

Описание стратегии