САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

На правах рукописи

Герасимов Александр Сергеевич

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМА ПОИСКА ВЫВОДА В РАСШИРЕНИИ БЕСКОНЕЧНОЗНАЧНОЙ ПРЕДИКАТНОЙ ЛОГИКИ ЛУКАСЕВИЧА

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

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель —
доктор физико-математических наук, профессор Н. К. Косовский

Санкт-Петербург – 2007

Оглавление

Глава 1.

Введение и предварительные сведения

В этой вводной главе приводится аннотация данной диссертационной работы, синтаксис и семантика бесконечнозначной предикатной логики Лукасевича, обзор работ, близких к этой диссертации по теме исследования, и обоснование актуальности направлений исследований, разрабатываемых в данной диссертации. Затем ставятся цели этой диссертации и излагается краткое содержание следующих глав.

Краткая характеристика работы. В данной работе предлагается логика более выразительная, чем бесконечнозначная предикатная логика Лукасевича. Язык предложенной логики расширяет язык логики Лукасевича так, что в предложенной логике выразимы различные модификаторы типа «очень», близкие к введенным Заде. Для предложенной логики сформулировано секвенциальное исчисление, аксиомы которого распознаются с помощью методов линейного программирования. Установлен ряд свойств этого исчисления, ориентированных на автоматизацию поиска логического вывода. Разработан алгоритм поиска вывода в предложенном исчислении. Доказаны свойства этого алгоритма, отражающие различные аспекты его корректности. Алгоритм поиска вывода реализован в виде интерфейса прикладного программирования на языке программирования Java. Доработан

и реализован алгоритм проверки совместности (и нахождения решения) систем линейных двучленных неравенств, который используется в качестве вспомогательного алгоритма для эффективного распознавания некоторых аксиом предложенного исчисления. Получена оценка временной сложности этого алгоритма в формальной вычислительной модели.

1.1. Актуальность решаемых задач и обзор близких работ

Интерес к многозначным (в частности, бесконечнозначным) логикам объясняется их разнообразными применениями, включающими представление нечетких знаний и приблизительные рассуждения (см., например, [58, 60]).

Первая многозначная (а именно, трехзначная пропозициональная) логика была предложена в работе Лукасевича [69]. В дальнейшем появились n-значные пропозициональные логики Лукасевича, бесконечнозначная пропозициональная логика Лукасевича, соответствующие им предикатные логики (или логики первого порядка) и другие многозначные логики. Логики Лукасевича до сих пор являются предметом активных исследований. Опишем бесконечнозначную предикатную логику Лукасевича (см., например, [62, 60, 58]), которая используется в данной работе.

1.1.1. Бесконечнозначная предикатная логика Лукасевича

Будем обозначать бесконечнозначную предикатную логику Лукасевича посредством Ł.

Термами являются предметные переменные и предметные константы.

Атомарные формулы имеют вид $P(t_1, \ldots, t_n)$, где P — n-местный предикатный символ, n — положительное целое число, $t_1, \ldots, t_n$ — термы.

Атомарные формулы и истинностные константы 0 и 1 являются фор-

мулами Ł. Если A и B — формулы Ł, x — предметная переменная, то $(A \to B)$, $\neg A$, $(A\&B)$, $(A \vee B)$, $\forall x A$ и $\exists x A$ являются формулами Ł.

Логические символы имеют следующие названия: импликация $\to$, отрицание $\neg$, конъюнкция $\&$, дизъюнкция $\vee$, квантор всеобщности $\forall$ и квантор существования $\exists$.

Пусть задана интерпретация I языка логики Ł, т.е.

1. задано непустое множество D,
2. каждой предметной константе сопоставлен элемент из D,
3. каждому n-местному предикатному символу сопоставлен предикат, действующий из D^n в отрезок действительных чисел $[0, 1]$.

Пусть также задана оценка v, т.е. отображение, сопоставляющее каждой предметной переменной элемент из D (тем самым свободным переменным в формулах присвоены значения из D). Тогда формула A получает истинностное значение $[A]_{I,v}$, являющееся действительным числом из отрезка $[0, 1]$, согласно правилам, приведенным после этого абзаца. Пусть w — оценка, x — предметная переменная, b — элемент из D; обозначим $w(b|x)$ такую оценку, что $w(b|x)(y) = w(y)$ для каждой предметной переменной y, отличной от x, и $w(b|x)(x) = b$.

1. $[(A \to B)]_{I,v} = \min(1, 1 - [A]_{I,v} + [B]_{I,v})$
2. $[\neg A]_{I,v} = 1 - [A]_{I,v}$
3. $[(A\&B)]_{I,v} = \min([A]_{I,v}, [B]_{I,v})$
4. $[(A \vee B)]_{I,v} = \max([A]_{I,v}, [B]_{I,v})$
5. $[\forall x A]_{I,v} = \inf\{[A]_{I,v(b|x)} | b \in D\}$
6. $[\exists x A]_{I,v} = \sup\{[A]_{I,v(b|x)} | b \in D\}$

Формула логики Ł называется *общезначимой*, если она принимает истинностное значение 1 во всякой интерпретации при любой оценке.

В логике Лукасевича выделяются, так сказать, «абсолютно истинные» формулы — формулы, принимающие истинностное значение 1 в любой интерпретации и при любой оценке. Однако полезно иметь средства для выражения «более или менее истинных» формул. Такие средства предоставляет предикатная рациональная логика Павелки [77, 60]. Эта логика расширяет бесконечнозначную предикатную логику Лукасевича путем добавления в язык истинностных констант, являющихся рациональными числами из отрезка $[0, 1]$.

Пусть A — формула, r — рациональное число из $[0, 1]$ и заданы интерпретация I и оценка v. Тогда $[A]_{I,v} \geq r$ если и только если $[(r \to A)]_{I,v} = 1$. Так выражаются различные степени истинности формулы.

Поскольку логика Лукасевича более известна, чем логика Павелки, ниже мы будем говорить о расширении логики Лукасевича, хотя предлагаемое в данной работе расширение бесконечнозначной предикатной логики Лукасевича является также и расширением предикатной рациональной логики Павелки.

1.1.2. Усиление выразительности логики Лукасевича

Многозначная логика как область математической логики развивалась параллельно с нечеткой логикой, восходящей к Заде [81, 82, 19]. Нечеткая логика Заде основана на теории нечетких множеств — множеств с размытыми границами; такое множество задается с помощью функции принадлежности, сопоставляющей элементу действительное число из отрезка $[0, 1]$. *Нечеткой логикой в широком смысле* (далее для краткости — нечеткой логикой) сейчас называют (см. [61]) дисциплину, использующую понятия теории нечетких множеств для разработки методов прикладных приблизительных рассуждений. Нечеткая логика используется в промышленных системах нечеткого контроля, например, в бытовых приборах. Однако с формально-логической точки зрения методы нечеткой логики не представляются корректно обоснованными [60].

В последнее десятилетие активно идет процесс формализации нечет-

кой логики (см. основополагающие труды [60, 58, 37, 29, 25]). В связи с этим *математической нечеткой логикой* (или *нечеткой логикой в узком смысле*) называют дисциплину, которая разрабатывает дедуктивные системы для нечеткой логики, рассматривая ее как многозначную логику, в стиле и со строгостью математической логики. Бесконечнозначная предикатная логика Лукасевича является одной из основных многозначных логик, используемых для формализации нечеткой логики.

Процесс становления математической нечеткой логики еще далек от завершения в связи с тем, что нечеткая логика включает концепции, которые отсутствуют в многозначной логике. В частности, в нечеткой логике имеются лингвистические модификаторы «очень», «чрезвычайно», «вполне» и др. Заде [19] применяет возведение функции принадлежности в квадрат для представления модификатора «очень», что делает формализацию нечеткой логики трудной. Потому альтернативная и более простая формализация существенной черты нечеткой логики — модификаторов типа «очень» — является важным шагом в сближении математической нечеткой логики с нечеткой логикой в широком смысле; к тому же это расширяет область возможных применений математической нечеткой логики.

1.1.3. Разработка методов и средств для автоматического доказательства в логике Лукасевича

Необходимым условием для многих успешных применений какой-либо логики является наличие удобных методов поиска доказательств в этой логике с помощью компьютера. Для формализации доказательств и их поиска используют исчисления.

Определение 1.1.3.1. Пусть задан язык (*язык исчисления*), слова которого будем называть объектами. *Исчисление* задает исходные объекты (*аксиомы*) и правила получения новых объектов из уже построенных (*правила вывода*) (см., например, [33, 39]).

Определение 1.1.3.2. *Выводом* (или *доказательством*) данного объекта в исчислении называется конечный список объектов, если последним элементом этого списка является данный объект, и каждый объект этого списка является аксиомой или получен из предшествующих объектов списка применением одного из правил вывода (иначе говоря, является *заключением* применения правила вывода к объектам-*посылкам*).

Определение 1.1.3.3. Объект называется *выводимым* в исчислении, если существует вывод этого объекта в данном исчислении.

К исчислению предъявим следующее естественное требование:

(DAR) разрешимость множества[1] аксиом и множества, состоящего из всех пар вида (конечный список объектов-посылок, объект-заключение), задаваемых всевозможными применениями правил вывода.

Замечание 1.1.3.4. Обычно это требование явно не высказывается, но из целей использования исчисления (надо хотя бы уметь решать, является ли заданный конечный список слов выводом или нет) это требование следует (ср. [57, раздел 4.1], где явно предполагается конечность числа видов, или схем, аксиом и правил вывода). Мы подчеркиваем это требование, поскольку ниже в этом разделе оно окажется существенным.

При рассмотрении исчисления для некоторой логики объектами могут быть формулы этой логики или более сложные конструкции, в некотором смысле соответствующие формулам. Применительно к описанной в разделе 1.1.1 логике Лукасевича задача интересующих нас исчислений состоит в том, чтобы в этих исчислениях были выводимы только общезначимые формулы; про такое исчисление говорят, что оно является *семантически обоснованным*. А исчисление, в котором выводима любая общезначимая формула, называют *полным* исчислением.

[1] Множество слов называют *разрешимым*, если существует алгоритм, выясняющий, принадлежит ли входное слово этому множеству или нет.

Базовые понятия теории алгоритмов предполагаются известными, хотя мы и поясняем некоторые из таких понятий. Детальнее с ними можно ознакомиться, например, по одной из книг [24, 43, 22, 23, 36].

1.1.3.1. Исчисления для бесконечнозначных логик Лукасевича

Для бесконечнозначной предикатной логики Лукасевича до этой работы были известны лишь исчисления гильбертовского типа (их можно найти, например, в [60, 58, 62]).

Общей чертой известных исчислений гильбертовского типа для рассматриваемой логики Лукасевича является наличие правила вывода модус поненс, которое выглядит так: из формул A и $(A \to B)$ получается формула B.

При поиске вывода заданной формулы B можно действовать следующим образом. Найти формулы, из которых формула B получается по одному из правил вывода, и для каждой найденной формулы, если она не является аксиомой, найти формулы, из которых она получается по одному из правил вывода и т.д. (Такой способ поиска вывода называется *поиском вывода снизу вверх* [33].)

Однако нахождение формул, из которых данная формула получается по правилу модус поненс, по сути превращается в угадывание, а по крайней мере одна из посылкой правила модус поненс является более длинной, чем заключение. По этой причине такие исчисления не подходят для автоматического поиска вывода заранее заданной формулы (см. также [57, разделы 4.1, 4.5]).

Для бесконечнозначной пропозициональной логики Лукасевича известны разнообразные методы поиска доказательств, в том числе разработанные в течение последних нескольких лет: резолюционные методы [80, 75], методы семантических таблиц [59, 76], секвенциальные исчисления [27, 78, 50, 53, 74, 73].

1.1.3.2. Уровневая логика и ее развитие

Особо отметим только что упомянутое секвенциальное исчисление [27] — исчисление для так называемой уровневой логики. В уровневой логике имеются следующие логические связки: конъюнкция и дизъюнкция, определенные как в логике Лукасевича (см. раздел 1.1.1), отрицание, соот-

ветствующее изменению знака истинностного значения, и бинарная связка, представляющая сложение истинностных значений. Эти связки позволяют записывать формулы логики Лукасевича. Аксиомы исчисления для уровневой логики определяются в терминах несовместности систем линейных неравенств и могут распознаваться с помощью методов линейного программирования, и этот подход развивается в данной работе.

Для логик с конечнозначными предикатами работа [27] развита в [29]: определены логики, секвенциальные исчисления для них и исследованы некоторые свойства этих исчислений, но проблемы автоматического поиска вывода не затронуты. В логиках конечнозначных предикатов [29] помимо связок уровневой логики есть унарная связка «весьма», которая соответствует удвоению истинностного значения. Однако представляется, что для выражения того, что некий объект «весьма» обладает некоторым свойством, нужно уменьшать, а не увеличивать положительное истинностное значение, соответствующее обладанию объекта свойством. Например, если свойству «Иван молод» соответствует истинностное значение 1/2, было бы естественно сопоставить свойству «Иван весьма молод» меньшее истинностное значение, скажем, 1/4. Таким образом, требуются более гибкие и выразительные логические средства (см. также раздел 1.1.2).

В диссертационной работе [44] описан и программно реализован алгоритм, который строит вывод в секвенциальном исчислении для подлогики смешанной логики Поста (смешанная логика Поста и секвенциальное исчисление для нее сформулированы в [29]). Но работа [44] не содержит теоретических оснований этого алгоритма и не затрагивает вопросы его корректности (среди которых должны быть, например, такие вопросы: «Действительно ли формула выводима (соответственно, невыводима), если алгоритм выдает ответ, что эта формула выводима (соответственно, невыводима)?»). Реализация этого алгоритма является самостоятельным приложением и не предоставляет программный интерфейс для доступа к своим возможностям, что сужает применимость этого приложения до его использования в диалоговом режиме с ручным вводом формул. Далее, правила

вывода секвенциального исчисления и порядок их применения «зашиты» в код, осуществляющий поиск вывода, что не позволяет изменить ни то, ни другое без переписывания большой части программы. Наконец, несмотря на то, что программа написана на языке Delphi, который поддерживает объектно-ориентированный стиль программирования,[2] в исходном коде этой программы нарушается концепция абстрактных типов данных,[3] что ярко иллюстрируется следующей строкой кода, приведенной в [44] на с. 94: «`pS^.inf.scv[Nscv+1].stk.Push(pS^.inf.scv[Nscv-1].stk.Pop(1));`». В общем, такой программный код практически не подлежит повторному использованию и настройке в соответствии с требованиями, сколь-нибудь отличающимися от тех, что имеются в виду в работе [44].

Для (конечнозначной) смешанной логики Поста, сформулированной в [29], был предложен метод резолюций [25], основанный на вложении смешанной логики Поста в двузначную логику.

Таким образом, настоящая диссертационная работа отчасти заполняет лакуну, образовавшуюся из-за того, что подход уровневой логики (распознавание аксиом секвенциального исчисления с помощью методов линейного программирования) не был развит для логик с бесконечнозначными предикатами, и не были исследованы проблемы автоматического поиска вывода для логик, восходящих к уровневой логике.

1.1.3.3. Заключительные замечания об исчислениях для бесконечнозначных логик Лукасевича

При попытках применения методов поиска доказательств в пропозициональной бесконечнозначной логике Лукасевича для вывода нечетких знаний ощущается ограниченность этих методов, поскольку они, в частности, не охватывают предикатную логику. Например, в [73] при представлении нечетких знаний приходится переводить предикатные формулы в пропозициональные, что, во-первых, можно осуществить только для конечных

[2] См., например, [8, 6, 45, 38].

[3] См. там же и, например, в [30, 20, 31].

областей определения предикатов и, во-вторых, значительно удлиняет запись формул, представляющих исходные знания.

Следует также отметить, что множество общезначимых формул бесконечнозначной предикатной логики Лукасевича не является перечислимым[4] (см. [79], а также [58, раздел 9.3.3]). Отсюда следует

Теорема 1.1.3.5. *Для бесконечнозначной предикатной логики Лукасевича не существует удовлетворяющее требованию (DAR), семантически обоснованное и полное исчисление.*

Доказательство. Если бы существовало такое исчисление, то можно было бы построить алгоритм E, которому подается на вход формула ϕ. Алгоритм E последовательно перебирает всевозможные конечные списки слов, проверяет с помощью алгоритма D, является ли очередной список выводом формулы ϕ в этом исчислении или нет; как только вывод формулы ϕ найден, выдается ответ «общезначима». Существование упомянутого алгоритма D, очевидно, обеспечивается разрешимостью множества аксиом исчисления и множества пар, задаваемых применениями правил вывода. Таким образом, алгоритм E выдает ответ «общезначима», если и только если входная формула общезначима, что противоречит неперечислимости множества общезначимых формул бесконечнозначной предикатной логики Лукасевича. □

Известные исчисления гильбертовского типа для бесконечнозначной предикатной логики Лукасевича неполны по указанной причине. Но все же исчисления для этой логики могут служить инструментами для получения многих общезначимых формул. По-видимому, такая ситуация напоминает ситуацию, связанную с неполнотой арифметики Пеано: общезначимая, но не выводимая в арифметике формула обычно искусственна, а задача изобретения такой естественной формулы очень трудна (см., например, [41]).

[4] Множество слов называют *перечислимым*, если существует алгоритм, который выдает ответ 1, если и только если входное слово принадлежит этому множеству (этот алгоритм может не заканчивать работу, если входное слово не принадлежит данному множеству). См. также сноску на с. 11.

Таким образом, неполнота исчислений для бесконечнозначной предикатной логики Лукасевича не является препятствием для исследований и применений различных исчислений (см. также [51, раздел 8]).

Итак, разработка удобных для поиска вывода исчислений для бесконечнозначной предикатной логики Лукасевича и реализация компьютерных программ для поиска вывода является перспективной исследовательской задачей. А возрастающий интерес к бесконечнозначной предикатной логике Лукасевича в связи с развитием математической нечеткой логики обусловливает потребность в автоматизации поиска доказательств для этой логики Лукасевича.

1.1.4. Реализация эффективного алгоритма решения систем линейных двучленных неравенств

Задача распознавания аксиомы секвенциального исчисления, предложенного в данной работе, сводится к проверке несовместности системы строгих и нестрогих линейных неравенств с целочисленными коэффициентами и рациональнозначными переменными. Особая роль систем линейных двучленных неравенств будет показана ниже в контексте вводимой логики и исчисления (см. раздел 2.4). Здесь же осветим общий контекст, в котором эффективный алгоритм решения систем линейных двучленных неравенств занимает важное место среди алгоритмов линейного программирования для решения систем линейных неравенств и является практически полезным. (Под алгоритмом решения системы понимается алгоритм, который либо выдает одно из решений системы, либо сообщает о несовместности системы.)

Рассмотрим задачу определения совместности (или решения) системы из m нестрогих линейных неравенств с целочисленными коэффициентами относительно n рациональнозначных переменных. Хорошо известно (см., например, [46, с. 191]), что полиномиальная разрешимость задачи линейного программирования эквивалентна полиномиальной разрешимости рассматриваемой задачи.

Для решения рассматриваемой задачи можно применить полиномиальные алгоритмы Л. Г. Хачияна [49] или Н. Кармаркара [68]. Число выполняемых этими алгоритмами арифметических операций зависит от величины коэффициентов системы.

Определение 1.1.4.1. Алгоритм называется *сильно полиномиальным*, если он полиномиален (по числу шагов при реализации на машине Тьюринга), и число элементарных арифметических операций (сложение, вычитание, умножение, деление, сравнение), выполняемых им над рациональными числами, ограничено полиномом от числа целых чисел на входе ([52], [40, с. 197]; ср. [72], [46, с. 304]).

Таким образом, число элементарных арифметических операций, выполняемых сильно полиномиальным алгоритмом, не зависит от величин целых чисел на входе.

Замечание 1.1.4.2. Отметим, что сильно полиномиальный алгоритм также называют истинно полиномиальным алгоритмом (см. [46, с. 304]). Однако в определении истинно полиномиального алгоритма из [46, с. 304] и [72] требование полиномиальности алгоритма явно не фигурирует. На наш взгляд, это существенное требование, выражающее, в частности, ограничение на возможную неэффективность организации структур данных, используемых в алгоритме.

До сих пор остается открытым вопрос о существовании сильно полиномиального алгоритма для определения совместности систем общего вида. Однако известны сильно полиномиальные алгоритмы для решения систем линейных неравенств с не более чем двумя переменными в каждом неравенстве, причем допускаются только нестрогие неравенства [72, 54, 63], На настоящий момент (ср. [52]) самым эффективным алгоритмом решения систем нестрогих линейных неравенств с не более чем двумя переменными в каждом неравенстве является алгоритм из работы [63] с $O(mn^2 \log m)$ арифметическими операциями.

В [18] предложен алгоритм проверки совместности (и опирающийся на него алгоритм решения) систем, с одной стороны, более узкого класса —

рассматриваются системы линейных двучленных неравенств (т.е. в каждом неравенстве не более двух переменных, и если в неравенство входят две переменные с ненулевыми коэффициентами, то свободный член в этом неравенстве нулевой), а с другой стороны, более широкого — допускаются строгие и нестрогие неравенства. Алгоритм основан на последовательном исключении переменных (см., например, [46, с. 239–242]) и удалении избыточных неравенств. В [18] показано, что алгоритм проверки совместности систем требует $O(n^3+m)$ элементарных арифметических операций над числами длиной $O(I)$, где I — длина входа. (Для алгоритма решения систем изменяется лишь длина чисел — $O(nI)$.)

Описание этого алгоритма проверки совместности систем не детализирует процесс удаления избыточных неравенств (хотя и содержит его идею). Также это описание не полностью определяет поведение алгоритма, поскольку пропущен один из случаев, когда после исключения очередной переменной полученная система оказывается пуста. Кроме того, этот (исправленный) алгоритм, строго говоря, не может быть назван сильно полиномиальным, пока не установлена его полиномиальность. В [18] дается лишь оценка числа элементарных арифметических операций над «длинными» числами, что не учитывает эффективность организации структур данных и не соответствует модели вычислений, близкой к компьютерам (ср. [4, глава 1]). Конечно, такая оценка допустима для абстрактного описания алгоритма, но для записи этого алгоритма на языке программирования и для оценки сложности алгоритма нужна большая степень детализации. Поэтому этот алгоритм нуждается в доработке, а оценка его сложности — учета реалистичной вычислительной модели и структур данных.

Наконец, этот алгоритм не был реализован на компьютере до данной работы. И вообще, нам не известны реализации упомянутых выше сильно полиномиальных алгоритмов. Распространенные системы компьютерной алгебры с богатыми возможностями, например, Mathematica [71], Maple [70], предлагают алгоритмы для более общих задач. Средствами Mathematica можно решать системы строгих и нестрогих линейных неравенств (кон-

кретный реализованный алгоритм не разглашается). Средствами Maple можно решать только системы нестрогих линейных неравенств с помощью симплекс-метода. Следует надеяться (и проверить это на практике), что для частной задачи решения систем строгих и нестрогих линейных двучленных неравенств алгоритм окажется более эффективным, чем, например, алгоритм, реализованный в Mathematica для общей задачи решения систем линейных неравенств.

Таким образом, доработка и реализация алгоритма решения систем строгих и нестрогих линейных двучленных неравенств имеет большую ценность.

1.2. Цели работы

Исходя из описанных выше направлений, требующих исследования и разработки, были поставлены следующие цели для данной работы.

- Разработка удобного для автоматического поиска вывода исчисления для бесконечнозначной предикатной логики Лукасевича, расширенной средствами для выражения модификаторов типа «очень».
- Разработка и реализация алгоритма поиска вывода в исчислении для расширения бесконечнозначной предикатной логики Лукасевича.
- Доработка, оценка временной сложности и реализация алгоритма решения систем линейных двучленных неравенств [18].

1.3. Краткое содержание работы

Вторая глава посвящена описанию предлагаемой логики (обозначаемой Lq), секвенциального исчисления LqS для нее и свойств этого исчисления [12, 11].

В первом разделе этой главы определяется язык логики Lq и его семантика. Неотъемлемой частью каждого предикатного символа является так

называемый его *отрезок истинностных значений* $[a, b]$, где a, b — рациональные числа, $a < b$. *Термом* является предметная переменная и предметная константа. *Атомарной формулой* является любое рациональное число, пропозициональная переменная и предикатный символ с заключенным в скобки списком термов. *Формулами* логики Lq являются атомарные формулы, а также $(A\&B)$, $(A \vee B)$, $(A \prec B)$, $q \cdot A$, $\forall x A$, $\exists x A$, где A и B — формулы логики Lq, q — рациональное число, x — предметная переменная. Связки $\prec$ и $q\cdot$ носят названия *нечеткое неравенство* и *модератор* соответственно.

Для задания семантики языка логики Lq вводятся понятия *интерпретации* и *оценки* языка. Эти понятия аналогичны традиционным, за исключением того, что здесь интерпретация сопоставляет предикатному символу предикат, который действует в отрезок действительных чисел, являющийся отрезком истинностных значений этого предикатного символа (каждой пропозициональной переменной сопоставляется действительное число из такого отрезка). Если заданы интерпретация и оценка, то каждая формула A получает *истинностное значение* $[A]$, являющееся действительным числом, согласно следующим правилам: $[(A\&B)] = \min([A], [B])$, $[(A \vee B)] = \max([A], [B])$, $[(A \prec B)] = [B] - [A]$, $[q \cdot A] = q \cdot [A]$, $[\forall x A] = \inf_x [A]$, $[\exists x A] = \sup_x [A]$. Формула называется *общезначимой*, если истинностное значение этой формулы неотрицательно во всякой интерпретации при любой оценке.

Отметим, что, во-первых, модераторы реализуют увеличение или уменьшение истинностных значений формул, поэтому с помощью модераторов можно формализовать различные модификаторы нечеткой логики типа «очень». Во-вторых, утверждение о том, что во всякой интерпретации при любой оценке формула A принимает истинностные значения, не меньшие (соответственно, не большие) рационального числа r, эквивалентно тому, что формула $(r \prec A)$ (соответственно, $(A \prec r)$) общезначима. В-третьих, любую формулу бесконечнозначной предикатной логики Лукасевича можно представить в виде формулы логики Lq так, что во всякой

интерпретации при любой оценке истинностные значения этих формул совпадают.

Приводится следующий пример формализации приблизительных рассуждений с помощью логики Lq. Из посылок (1) если предмет мал, то его трудно разглядеть, и (2) предмет z очень мал, следует, что (3) предмет z довольно трудно разглядеть. Введем предикат $P1[0,1](x)$, который сопоставляет предмету x действительное число из отрезка $[0,1]$, характеризующее степень малости этого предмета, и предикат $P2[0,1](y)$, аналогичным образом выражающий степень трудности разглядывания предмета y. Модификатор «очень» формализуем с помощью модератора $1/2\cdot$, «довольно» — с помощью модератора $2/3\cdot$. Указанное приблизительное рассуждение запишем в виде формулы логики Lq: $((\forall x(P1[0,1](x) \prec P2[0,1](x)) \,\&\, 1/2 \cdot P1[0,1](z)) \prec 2/3 \cdot P2[0,1](z))$. Таким образом, для обоснования этого приблизительного рассуждения достаточно доказать соответствующую ему формулу.

В конце первого раздела второй главы доказана теорема о том, что множество общезначимых формул логики Lq неперечислимо.

Во втором разделе второй главы сформулировано безантецедентное секвенциальное исчисление LqS для логики Lq, и определены сопутствующие понятия. *Секвенцией* называется конечный список формул логики Lq (*членов* этой секвенции), разделенных запятыми; некоторые формулы могут повторяться; порядок формул в списке не имеет значения. Секвенция называется *общезначимой*, если дизъюнкция всех ее членов общезначима (пустая секвенция в виде такой дизъюнкции представляется числом -1). Приводятся *правила вывода*, вводящие логические символы, кроме модератора перед атомарной формулой и нечеткого неравенства (формулы, состоящие из атомарных формул с модераторами и нечетких неравенств, обрабатываются при распознавании аксиом).

Далее определены аксиомы исчисления LqS. *Каноническая цепь неравенств (КЦН)* определяется следующим образом. Формулы вида P и $q \cdot P$ (где $q\cdot$ — модератор, P — атомарная формула) являются КЦН. Если I и

J — КЦН, то $(I \prec J)$ является КЦН.

Пусть дана секвенция S. Удалим из S все члены, которые не являются КЦН; тогда полученная секвенция называется *базовой подсеквенцией* секвенции S. Секвенция называется *аксиомой*, если базовая подсеквенция этой секвенции общезначима.

Любой секвенции с непустой базовой подсеквенцией сопоставляется (с помощью приведенного в тексте диссертации алгоритма) система строгих и нестрогих линейных неравенств с рациональными коэффициентами и рациональнозначными переменными. Доказана теорема о том, что такая секвенция является аксиомой, если и только если соответствующая система линейных неравенств несовместна.

В третьем разделе второй главы устанавливаются некоторые свойства исчисления LqS. Перечислим наиболее важные из них.

1. Все правила вывода и обратные к ним сохраняют общезначимость секвенций. Исчисление является семантически обоснованным.

2. Исчисление непротиворечиво.

3. Безантецедентное секвенциальное исчисление для классической двузначной логики вкладывается в исчисление LqS.

4. Для логики Lq не существует полное и семантически обоснованное исчисление.

5. Исчисление LqS неразрешимо.

6. Исчисление LqS полно для пропозиционального фрагмента логики Lq.

7. Пропозициональный фрагмент исчисления LqS разрешим.

8. Исчисление LqS допускает минус-нормализацию вывода.

Поясним последнее свойство. При так называемом поиске вывода заданной секвенции S *снизу вверх* находят секвенции, из которых секвенция S получается по одному из правил вывода, и для каждой найденной секвенции, если она не является аксиомой, находят секвенции, из которых она получается по одному из правил вывода, и т.д. При этом по секвенции S и правилу вывода требуется подбирать секвенции, которые явля-

ются посылками применения этого правила вывода, причем секвенция S должна быть заключением этого применения (т.е. требуется осуществлять *контрприменение* правила вывода). При контрприменении правил вывода исчисления LqS, вводящих какой-либо выбранный логический символ, секвенции-посылки подбираются детерминированным образом, кроме того случая, когда осуществляется контрприменение так называемого *минус-правила*. Каждое из минус-правил исчисления LqS (типа правила введения квантора существования в сукцедент секвенции в генценовском исчислении для классической двузначной логики) допускает бесконечный перебор термов для подстановки в посылку при контрприменении.

Для классической двузначной логики первого порядка известны секвенциальные исчисления (см., например, [21], а также [34, 55]), в которых устранен бесконечный перебор термов для подстановки при контрприменениях минус-правил соответствующих секвенциальных исчислений. Такое устранение бесконечного перебора термов названо *минус-нормализацией* [34]. Однако упомянутые работы (и все известные нам другие работы) не содержат доказательств равнообъемности предложенных секвенциальных исчислений и какого-либо из традиционных секвенциальных исчислений для классической двузначной логики.

В настоящей работе формулируется ограничение на термы для подстановки (подставляемый терм — один из конечного числа термов, входящих в заключение минус-правила) и доказывается, что при этом не изменяется объем выводимых секвенций. (Тогда обоснование минус-нормализации для исчисления классической двузначной логики следует из свойства 3.)

В последнем, четвертом разделе второй главы формулируется подлогика $Lq2$ логики Lq и секвенциальное исчисление $Lq2S$ для этой подлогики. В формулах подлогики $Lq2$ использование логической связки $\prec$ ограничено так, что распознавание аксиомы исчисления $Lq2S$ сводится к проверке несовместности системы линейных неравенств, каждое из которых имеет не более двух членов. Для решения таких задач существует сильно полиномиальный алгоритм (такой алгоритм описан в пятой главе данной работы),

тогда как для распознавания аксиом исчисления LqS известны только полиномиальные алгоритмы. Подлогика $Lq2$ оказывается достаточно выразительной для первоначального моделирования областей нечетких знаний.

В **третьей главе** описан алгоритм поиска вывода и доказаны его свойства.

При поиске вывода заданной секвенции в исчислении LqS снизу вверх естественным образом строится *дерево поиска вывода*, корнем которого является исходная секвенция, непосредственными потомками корня — секвенции, являющиеся посылками контрприменения некоторого правила вывода к корневой секвенции, и т.д. (считается, что это дерево растет от корня вверх). В тот момент, когда все секвенции-листья оказываются аксиомами, дерево поиска вывода становится *деревом вывода*, т.е. вывод найден.

Несмотря на то, что бесконечный перебор термов для подстановки при контрприменениях минус-правил может быть устранен, задача подбора термов, хотя и упростилась, все равно остается. Воспользуемся так называемым *методом метапеременных*: вместо подстановки конкретного терма при контрприменении минус-правила отложим выбор терма, подставив вместо него уникальную метапеременную. Тем самым вместо дерева поиска вывода строится так называемая *заготовка вывода*, содержащая метапеременные. Тогда в ходе поиска вывода время от времени нужно проверять, можно ли подобрать термы-значения для метапеременных так, чтобы превратить заготовку вывода в дерево вывода.

Предлагаемый алгоритм *Prove* [13] ищет вывод заданной секвенции, строя заготовку вывода. При контрприменении минус-правила и введении метапеременной с ней ассоциируется *подстановочное множество* — конечное множество термов, содержащее все термы, только которые и достаточно подставлять вместо этой метапеременной в соответствии с ограничением минус-нормализации. Тогда для того, чтобы подобрать значения для метапеременных, превращающие заготовку вывода в дерево вывода, производится *унификация* — конечный перебор термов из подстановочных множеств метапеременных, входящих в заготовку вывода, и при этом пе-

реборе выбираются значения метапеременных, при которых заготовка превращается в дерево вывода.

В первом разделе третьей главы обсуждается идея алгоритма и определяются используемые понятия.

Во втором разделе описаны шаги главного алгоритма *Prove* и вспомогательных алгоритмов: алгоритма, проверяющего, является ли секвенция аксиомой, и алгоритма, проводящего унификацию. Также сформулированы требования к вспомогательному алгоритму, называемому *тактикой поиска вывода*: такой алгоритм по заготовке вывода сообщает, когда нужно провести унификацию, и выбирает (a) секвенцию-лист S, (b) правило вывода (R), контрприменение которого следует осуществить, и (c) вхождение логического символа в S, которое может быть введено в S применением правила (R). В конце второго раздела приводится пример поиска вывода с комментариями.

В третьем разделе доказаны свойства вспомогательных алгоритмов и главного алгоритма *Prove*. Итогом является следующая теорема.

Теорема. *Пусть алгоритм Prove использует любую тактику поиска вывода; S — секвенция, подаваемая на вход алгоритма Prove. Тогда верны следующие утверждения.*

(1) Если алгоритм Prove выдал ответ «выводима», то секвенция S выводима в исчислении LqS.

(2) Если алгоритм Prove выдал ответ «невыводима», то секвенция S невыводима в исчислении LqS.

(3) Пусть секвенция S не содержит кванторов. Тогда алгоритм Prove выдает ответ «выводима», если секвенция S выводима в исчислении LqS, и выдает ответ «невыводима», если секвенция S невыводима в исчислении LqS.

В конце третьего раздела обсуждается выбор тактики поиска вывода, и описывается алгоритм-тактика, который выбирает минус-правила для

контрприменения «равномерно», давая равные возможности разным вхождениям кванторов участвовать в контрприменениях.

В **четвертой главе** описана программная реализация алгоритма поиска вывода [13]. Этот алгоритм реализован на языке программирования Java в виде интерфейса прикладного программирования (ИПП), который предоставляет программный интерфейс для доступа к своим функциональным возможностям. В начале этой главы кратко описывается назначение основных классов, сгруппированных в пакеты, и приводятся диаграммы классов, изображающие иерархии логических символов, термов и формул.

Затем определяется политика разделения объектов в программном представлении формул (с помощью синтаксических деревьев с возможно разделяемыми листьями) и секвенций. Эта политика, с одной стороны, позволяет однозначно и эффективно идентифицировать вхождение неатомарной подформулы в формулу, а, с другой стороны, позволяет существенно экономить память при поиске вывода, допуская общие формулы в разных секвенциях.

Далее описываются основные компоненты реализации алгоритма поиска вывода. Отметим следующие ключевые особенности.

В этом ИПП тактика поиска вывода задается с помощью интерфейса `Tactics`. Главный алгоритм поиска вывода может использовать любую тактику, являющуюся реализацией этого интерфейса. (Такая архитектура соответствует образцу проектирования *стратегия* [42].)

Каждое правило вывода представлено объектом, который имеет метод, осуществляющий контрприменение этого правила. Главный алгоритм поиска вывода делегирует контрприменение правила вывода, выбранного тактикой, самому этому правилу. Таким образом, правила вывода могут быть легко изменены.

Каждая система линейных неравенств, получаемая при распознавании аксиом, проверяется на несовместность вспомогательным алгоритмом, описанным в пятой главе данной работы, если все неравенства системы двучленные, иначе — функцией `FindInstance` системы компьютерной алгебры

Mathematica (посредством J/Link — инструментария, связывающего программу на Java с Mathematica).

В конце четвертой главы приведен систематический обзор предоставляемого программного интерфейса для поиска вывода.

Объем написанного программного кода, включая реализацию алгоритма решения систем линейных двучленных неравенств, составляет около 9000 строк.

Пятая глава посвящена алгоритму проверки совместности систем строгих и нестрогих линейных двучленных неравенств с целочисленными коэффициентами и рациональнозначными переменными, а также алгоритму решения таких систем (под алгоритмом решения системы понимается алгоритм, находящий хотя бы одно решение системы, если оно существует, и сообщающий о несовместности системы в противном случае). Эти алгоритмы основаны на методе исключений переменных и одновременном с исключением переменной удалении избыточных неравенств так, что количество неравенств в системе, содержащих любые две переменные, не превосходит заранее фиксированную константу.

Упомянутые алгоритмы были предложены в [18]. В настоящей диссертационной работе осуществлена детализация этих алгоритмов, причем добавлены необходимые шаги (без которых описания алгоритмов из [18] были некорректны), и разработан вспомогательный алгоритм удаления избыточных неравенств (см. [15, 16]). Доказана корректность описанных в настоящей диссертации алгоритмов, т.е. алгоритм проверки совместности систем выдает ответ «совместна», если входная система совместна, и выдает ответ «несовместна», если входная система несовместна (аналогичное утверждение установлено и для алгоритма решения систем).

Далее определена вычислительная модель, близкая к равнодоступной адресной машине с логарифмическим весовым критерием (см. [4]), и получена полиномиальная оценка временной сложности описанных алгоритмов [16]. Установлена полиномиальность числа шагов этих алгоритмов при реализации на машине Тьюринга, что позволяет также доказать их силь-

ную полиномиальность (см. раздел 1.1.4).

В последнем разделе пятой главы описана реализация алгоритма на языке Java в виде интерфейса прикладного программирования [14]. В этой реализации участвуют 2 алгоритма с похожими шагами: упомянутый алгоритм решения систем с удалением избыточных неравенств и алгоритм, основанный на обычном методе исключений переменных. Одна из основных задач объектно-ориентированного программирования — вычленение общего поведения с целью повторного использования кода — решена с помощью *шаблонного метода* [42].

Наконец, приводятся результаты экспериментов по сравнению производительности реализованного автором диссертации алгоритма решения систем и алгоритма, представленного функцией `FindInstance` системы компьютерной алгебры Mathematica. Эксперименты проводились на одном и том же персональном компьютере. На решение систем из нескольких тысяч неравенств реализованный алгоритм затрачивает несколько секунд, а Mathematica — несколько десятков минут. На решение систем из нескольких десятков тысяч неравенств реализованный алгоритм затрачивает до нескольких десятков секунд, а Mathematica не заканчивает работу за 12 часов. Эти результаты говорят о том, что для решения систем линейных двучленных неравенств реализованный алгоритм намного эффективнее алгоритма, используемого в системе компьютерной алгебры Mathematica.

Шестая глава содержит список основных результатов, полученных в данной работе.

В **приложении А** приводится исходный текст небольшой программы, использующей разработанный ИПП для поиска вывода. Приведенный в следующем разделе этого приложения протокол поиска вывода формулы получен с помощью этой программы. В последнем разделе приложения А описан пример формализации нечетких знаний, заданных предложениями естественного языка, с помощью логики Lq и вывода новых нечетких знаний из заданных.

Приложение Б содержит исходный текст небольшой программы, ис-

пользующей разработанный ИПП для решения систем линейных двучленных неравенств, а также служащей для сравнения производительности реализованного автором диссертации алгоритма решения систем и алгоритма, представленного функцией `FindInstance` системы компьютерной алгебры Mathematica.

Глава 2.

Расширение бесконечнозначной предикатной логики Лукасевича

В данной главе определен язык предлагаемой логики Lq и его семантика. Установлена семантическая связь логики Lq с бесконечнозначной предикатной логикой Лукасевича, и доказана неперечислимость общезначимых формул логики Lq. Сформулировано секвенциальное исчисление LqS для предложенной логики и доказан ряд свойств, среди которых семантическая обоснованность и непротиворечивость исчисления, вложение секвенциального исчисления для классической двузначной логики в исчисление LqS, полнота и разрешимость пропозиционального фрагмента предложенного исчисления, устранение бесконечного перебора термов при поиске вывода снизу вверх.

2.1. Язык логики и его семантика

2.1.1. Язык логики

Обозначение 2.1.1.1. Предлагаемую логику обозначаем посредством Lq.

Определение 2.1.1.2. *Язык логики* Lq задается набором из трех множеств:

1. множества *предметных переменных*, которые являются любыми словами из букв английского алфавита и десятичных цифр, причем такие слова начинаются со строчной буквы;

2. множества *предметных констант*, которые обозначаются словами вида $\#\alpha$, где α — любое слово из букв английского алфавита и десятичных цифр;

3. непустого множества *предикатных символов*, которые обозначаются словами вида $P\alpha[a, b]$, где α — любое слово из букв английского алфавита и десятичных цифр, a, b — любые рациональные числа, $a < b$. С каждым предикатным символом связано неотрицательное целое число — его местность. 0-местный предикатный символ также называют *пропозициональной переменной*. Отрезок $[a, b]$ называется *отрезком истинностных значений* предикатного символа (пропозициональной переменной) $P\alpha[a, b]$. Два предикатных символа $P\alpha_1[a_1, b_1]$ и $P\alpha_2[a_2, b_2]$ считаются одинаковыми, если они имеют одинаковую местность, α_1 и α_2 равны как слова, a_1 и a_2 равны как рациональные числа, b_1 и b_2 равны как рациональные числа.

Определим формулы логики Lq.

Определение 2.1.1.3. *Термом* является предметная переменная и предметная константа.

Определение 2.1.1.4. *Атомарной формулой* является любое рациональное число и пропозициональная переменная. Если P — n-местный преди-

катный символ, $n > 0$, $t_1, \ldots, t_n$ — термы, то $P(t_1, \ldots, t_n)$ также является атомарной формулой.[1]

Для построения формул используются следующие *логические символы*: *логические связки* $\&$ (конъюнкция), $\vee$ (дизъюнкция), $\prec$ (нечеткое неравенство), $q\cdot$ (модератор), где q — рациональное число, и *кванторы* $\forall$ (квантор всеобщности), $\exists$ (квантор существования).

Определение 2.1.1.5. Атомарная формула является *формулой логики* Lq. Если A и B — формулы, q — рациональное число, x — предметная переменная, то $(A\&B)$, $(A \vee B)$, $\forall xA$, $\exists xA$, $(A \prec B)$, $q \cdot A$ являются формулами логики Lq.

Предполагается известной терминология, связанная с синтаксисом языков первого порядка (в частности, подформула, свободное и связанное вхождение предметной переменной, подстановка терма в формулу, свобода для подстановки). Эту терминологию можно найти в одном из учебников по математической логике, например, [24, 22, 23, 36, 29]. Здесь же приведем не столь распространенную терминологию.

Определение 2.1.1.6. В формулах вида $\forall xA$, $\exists xA$ выражение $\forall x$ или $\exists x$ называется *кванторной приставкой*.

Определение 2.1.1.7. *Деревом формулы* A назовем упорядоченное дерево, корень которого помечен (или является)

- самой формулой A, если A атомарна;
- логической связкой β, если A имеет вид βB, причем в этом случае непосредственным потомком этого узла является дерево формулы B;
- логической связкой γ, если A имеет вид $B\gamma C$, причем в этом случае непосредственными потомками этого узла являются деревья формул B и C;

[1] Местность предикатного символа не входит в его обозначение, поскольку она легко определяется по числу термов-аргументов этого предикатного символа.

- кванторной приставкой δ, если A имеет вид δB, причем в этом случае непосредственным потомком этого узла является дерево формулы B.

2.1.2. Семантика

Определение 2.1.2.1. Будем говорить, что задана *интерпретация* языка логики Lq, если

1. задано непустое множество D, называемое *носителем*;
2. каждой предметной константе сопоставлен элемент из D;
3. каждому n-местному предикатному символу сопоставлен предикат, который действует из D^n в отрезок действительных чисел $[a, b]$, являющийся отрезком истинностных значений этого предикатного символа (в частном случае $n = 0$ каждой пропозициональной переменной сопоставлено действительное число из этого отрезка).

Определение 2.1.2.2. Если задана интерпретация языка, *оценкой* языка называется функция из множества предметных переменных в носитель.

Замечание 2.1.2.3. Оценка служит для задания свободным переменным в формулах значений из D.

Обозначение 2.1.2.4. Пусть задана интерпретация I с носителем D. Далее, пусть w — оценка, x — предметная переменная, b — элемент из D. Обозначим $w(b|x)$ такую оценку, что $w(b|x)(y) = w(y)$ для каждой предметной переменной y, отличной от x, и $w(b|x)(x) = b$.

Определение 2.1.2.5. Пусть заданы интерпретация I с носителем D и оценка v языка. Тогда каждая формула A получает *истинностное значение* $[A]_{I,v}$, являющееся действительным числом, согласно следующим правилам.

1. $[(A \prec B)]_{I,v} = [B]_{I,v} - [A]_{I,v}$

2. $[q \cdot A]_{I,v} = q \cdot [A]_{I,v}$

3. $[(A\&B)]_{I,v} = \min([A]_{I,v}, [B]_{I,v})$

4. $[(A \vee B)]_{I,v} = \max([A]_{I,v}, [B]_{I,v})$

5. $[\forall x A]_{I,v} = \inf\{[A]_{I,v(b|x)} | b \in D\}$

6. $[\exists x A]_{I,v} = \sup\{[A]_{I,v(b|x)} | b \in D\}$

Замечание 2.1.2.6. В некоторых случаях, когда рассматривается одна фиксированная интерпретация и одна фиксированная оценка, истинностное значение формулы будем обозначать $[A]$.

Определение 2.1.2.7. Назовем формулу *общезначимой*, если истинностное значение этой формулы неотрицательно во всякой интерпретации при любой оценке.

Замечание 2.1.2.8. Нечеткое неравенство можно рассматривать как многозначную импликацию. Оно сходно с импликацией Лукасевича в том, что из одного истинностного значения x следует не меньшее истинностное значение y. Точнее, $(x \prec y)$ общезначима в логике Lq тогда и только тогда, когда $x \leq y$; и $(x \to y)$ общезначима в логике Лукасевича тогда и только тогда, когда $x \leq y$.

Замечание 2.1.2.9. Модераторы реализуют увеличение или уменьшение истинностных значений формул. Поэтому с помощью модераторов можно формализовать различные модификаторы нечеткой логики типа «очень» (см. раздел 1.1.2). Модератор $-1\cdot$ трактуется как отрицание.

Пример 2.1.2.10. Приведем пример формализации приблизительных рассуждений с помощью логики Lq. Из посылок (1) если предмет мал, то его трудно разглядеть, и (2) предмет z очень мал, следует, что (3) предмет z довольно трудно разглядеть. Введем предикат $P1[0,1](x)$, который сопоставляет предмету x действительное число из отрезка $[0,1]$, характеризующее степень малости этого предмета, и предикат $P2[0,1](y)$,

аналогичным образом выражающий степень трудности разглядывания предмета y. Модификатор «очень» формализуем с помощью модератора $1/2\cdot$, «довольно» — с помощью модератора $2/3\cdot$. Указанное приблизительное рассуждение запишем в виде формулы логики Lq: $((\forall x(P1[0,1](x) \prec P2[0,1](x)) \,\&\, 1/2 \cdot P1[0,1](z)) \prec 2/3 \cdot P2[0,1](z))$. Таким образом, для обоснования этого приблизительного рассуждения достаточно доказать соответствующую ему формулу.

Замечание 2.1.2.11. Ограниченность истинностных значений каждого предиката влечет конечность истинностного значения любой формулы, в частности, формулы с кванторами.

Замечание 2.1.2.12. В определении общезначимой формулы логики Lq выделено множество истинностных значений — множество неотрицательных действительных чисел. В принципе, можно выделить другие множества истинностных значений, но наиболее употребительные случаи могут быть выражены в терминах выделенного нами множества истинностных значений. Именно, утверждение о том, что во всякой интерпретации при любой оценке формула A принимает истинностные значения, не меньшие (соответственно, не большие) рационального числа r, эквивалентно тому, что формула $(r \prec A)$ (соответственно, $(A \prec r)$) общезначима.

Замечание 2.1.2.13. Интерпретацию языка можно было бы определить так, что отрезки истинностных значений состояли бы лишь из рациональных чисел. Тогда для пропозиционального фрагмента логики Lq истинностные значения были бы рациональными числами; но для (предикатной) логики Lq истинностные значения все равно были бы, вообще говоря, действительными числами, поскольку множество рациональных чисел не замкнуто относительно операций inf и sup.

2.1.3. Соотношение с логикой Лукасевича

Определение 2.1.3.1. Пусть A — формула бесконечнозначной предикатной логики Лукасевича B — формула логики Lq, причем отрезок истин-

ностных значений каждого предикатного символа формулы B есть $[0,1]$. Кроме того, любой предикатный символ P логики Лукасевича отождествим с предикатным символом $P[0,1]$ логики Lq. Тогда назовем формулы A и B *эквивалентными*, если во всякой интерпретации при любой оценке истинностное значение формулы A совпадает с истинностным значением формулы B .

Предложение 2.1.3.2 (Соотношение логики Lq с логикой Лукасевича)**.** *Для любой формулы ϕ бесконечнозначной предикатной логики Лукасевича найдется эквивалентная ей формула ψ логики Lq.*

Доказательство. Истинностные значения отрицания и импликации Лукасевича вычисляются следующим образом: $[\neg A] = 1 - [A]$ и $[(A \to B)] = \min(1 + [B] - [A], 1)$. Поэтому формулы логики Лукасевича вида $\neg A$ и $(A \to B)$ представляются в виде эквивалентных формул логики Lq как $(A \prec 1)$ и $(((1 \prec A) \prec B)\&1)$. Истинностные значения для остальных логических символов логики Лукасевича вычисляются так же, как и в логике Lq.

Припишем к каждому предикатному символу формулы ϕ отрезок истинностных значений $[0,1]$. В формуле ϕ заменим все подформулы вида $\neg A$ и $(A \to B)$ на приведенные в предыдущем абзаце эквивалентные формулы логики Lq и получим искомую формулу ψ. □

Таким образом, логику Lq можно рассматривать как расширение бесконечнозначной предикатной логики Лукасевича.

Теорема 2.1.3.3. *Множество общезначимых формул логики Lq не является перечислимым.*[2]

Доказательство. Предположим, что множество общезначимых формул логики Lq перечислимо. Тогда существует алгоритм f, который для любой формулы A логики Lq выдает ответ 1, если и только если A

[2]См. сноску 4 на с. 15.

общезначима (в соответствии с определением общезначимости для логики Lq).

Построим следующий алгоритм g, на вход которому подается формула B бесконечнозначной предикатной логики Лукасевича:

1) преобразовать B в эквивалентную формулу A логики Lq (см. предложение 2.1.3.2);

2) заменить A на формулу $(1 \prec A)$, обозначаемую A';

3) выдать результат работы $f(A')$.

Алгоритм g для любой формулы B бесконечнозначной предикатной логики Лукасевича выдает ответ 1, если и только если B общезначима (в соответствии с определением общезначимости для логики Лукасевича). Следовательно, множество общезначимых формул бесконечнозначной предикатной логики Лукасевича перечислимо. Однако это не так (см. [79], а также конец раздела 1.1.3). Пришли к противоречию, поэтому множество общезначимых формул логики Lq не является перечислимым. □

2.2. Секвенциальное исчисление

Для логики Lq предлагается (безантецедентное) секвенциальное исчисление, обозначаемое LqS.

Определение 2.2.0.4. *Секвенцией* называется конечный список формул логики Lq, разделенных запятыми; некоторые формулы могут повторяться; порядок формул в списке не имеет значения. Этот список формул будем также называть *сукцедентом* секвенции. Каждую из формул этого списка назовем *членом* секвенции (или ее сукцедента).

Определение 2.2.0.5. Назовем секвенцию *чистой*, если

(1) никакая предметная переменная не входит в эту секвенцию свободно и связанно одновременно, и

(2) никакая предметная переменная не связана вхождением квантора, находящимся в области действия другого вхождения квантора, которое связывает ту же самую переменную.

Замечание 2.2.0.6. Заметим, что это определение чистоты находится «посредине» между определением из [29, с. 43], где требуется лишь (1), и определением из [24, с. 68], где (2) заменено на более сильное требование: любые два различные вхождения кванторных приставок связывают различные переменные. Как мы увидим в дальнейшем, данное нами определение чистоты удобно для автоматического поиска вывода.

Аналогично можно говорить о чистом списке формул и о чистой формуле.

Потребуем, чтобы всякая секвенция исчисления LqS была чистой.

Определение 2.2.0.7. *Секвенция (или конечный список формул)* Δ *представляется в виде формулы* $\Phi(\Delta)$ как дизъюнкция всех формул этой секвенции (списка). Представлением пустой секвенции (пустого списка формул), т.е. секвенции, не содержащей ни одной формулы, в виде формулы является отрицательное число -1.

Определение 2.2.0.8. Назовем секвенцию *общезначимой*, если ее представление в виде формулы является общезначимой формулой.

Язык исчисления LqS составляют всевозможные секвенции. Определения *вывода секвенции* и *выводимой секвенции* являются частными случаями определений 1.1.3.2 и 1.1.3.3.

Определение 2.2.0.9. Формула называется *выводимой*, если секвенция, состоящая из одной этой формулы, выводима.

Замечание 2.2.0.10. В дальнейшем для сокращения записи будем иногда отождествлять формулу и секвенцию, состоящую из одной этой формулы.

2.2.1. Правила вывода

Для формулировки правил вывода исчисления LqS понадобятся следующие обозначения и определения.

Ниже F_1 и F_2 обозначают части некоторой формулы, так что F_1FF_2 (т.е. результат конкатенации цепочек символов F_1, F, F_2) является формулой для некоторой выделенной формулы F.

Определение 2.2.1.1. *Знак (положительный или отрицательный) вхождения связки $\prec$ в формулу* определяется следующим образом. Подчеркнутое вхождение $\prec$ в формулу вида $(G\underline{\prec}H)$ является положительным. Если подчеркнутое вхождение $\prec$ в формулу вида $F_1(G\underline{\prec}H)F_2$ является положительным (соответственно, отрицательным), то подчеркнутое вхождение $\prec$ в формулу вида $(F_1(G\underline{\prec}H)F_2 \prec I)$ является отрицательным (соответственно, положительным), и подчеркнутое вхождение $\prec$ в формулу вида $(I \prec F_1(G\underline{\prec}H)F_2)$ является положительным (соответственно, отрицательным). В других случаях знак вхождения $\prec$ не определен.

Обозначение 2.2.1.2. Там, где это необходимо, обозначаем положительное вхождение $\prec$ как $\prec_+$, отрицательное вхождение $\prec$ как $\prec_-$; если знак вхождения $\prec$ не определен или не важен, используем $\prec$.

Пример 2.2.1.3. $(((G\&(H \prec I)) \prec_- J) \prec_+ K)$, причем знак первого вхождения $\prec$ в эту формулу не определен.

Замечание 2.2.1.4. Если формула представлена в виде дерева (см. определение 2.1.1.7 на с. 32), то знаки вхождений связки $\prec$ легко расставляются следующим образом. Если корень дерева формулы помечен $\prec$, то это вхождение $\prec$ положительное. Далее, при каждом спуске от узла к его правому непосредственному потомку, который помечен $\prec$, знак вхождения сохраняется; при каждом спуске от узла к его левому непосредственному потомку, который помечен $\prec$, знак вхождения меняется на противоположный. Если при таком спуске встретился узел, не помеченный $\prec$, то далее нет смысла спускаться: знаки вхождений $\prec$ ниже не определены.

Обозначение 2.2.1.5. Для формулировки правил вывода примем следующие обозначения. Пусть q, q_1, q_2 — рациональные числа, такие что $q = q_1 \cdot q_2$, q_p — неотрицательное рациональное число, q_n — отрицательное рациональное число; Δ — список формул (может, пустой); A, B, C — формулы; x, y — любые предметные переменные, t — некоторый терм; $A(t|x)$ — результат подстановки терма t вместо всех свободных вхождений предметной переменной x в A.

Используются следующие правила вывода. Справа от каждого правила вывода указано его обозначение.

Правила вывода, отчасти аналогичные правилам вывода безантецедентного секвенциального исчисления для классической двузначной логики:

$$\frac{\Delta, q \cdot A}{\Delta, q_1 \cdot q_2 \cdot A}\ (q_1 \cdot q_2), \qquad \frac{\Delta, q_p \cdot A \quad \Delta, q_p \cdot B}{\Delta, q_p \cdot (A \& B)}\ (q_p\&), \qquad \frac{\Delta, q_n \cdot A \quad \Delta, q_n \cdot B}{\Delta, q_n \cdot (A \vee B)}\ (q_n\vee),$$

$$\frac{\Delta, q_n \cdot A, q_n \cdot B}{\Delta, q_n \cdot (A \& B)}\ (q_n\&), \quad \frac{\Delta, q_p \cdot A, q_p \cdot B}{\Delta, q_p \cdot (A \vee B)}\ (q_p\vee), \quad \frac{\Delta, q_p \cdot A(t|x), q_p \cdot \exists x A}{\Delta, q_p \cdot \exists x A}\ (q_p\exists),$$

$$\frac{\Delta, q_p \cdot A(y|x)}{\Delta, q_p \cdot \forall x A}\ (q_p\forall), \qquad \frac{\Delta, q_n \cdot A(y|x)}{\Delta, q_n \cdot \exists x A}\ (q_n\exists), \qquad \frac{\Delta, q_n \cdot A(t|x), q_n \cdot \forall x A}{\Delta, q_n \cdot \forall x A}\ (q_n\forall).$$

Правила вынесения модератора из нечеткого неравенства:

$$\frac{\Delta, F_1(q_p \cdot A \prec' q_p \cdot B)F_2}{\Delta, F_1 q_p \cdot (A \prec B)F_2}\ (q_p \prec), \qquad \frac{\Delta, F_1(q_n \cdot A \prec' q_n \cdot B)F_2}{\Delta, F_1 q_n \cdot (B \prec A)F_2}\ (q_n \prec),$$

где $\prec'$ есть $\prec_+$ или $\prec_-$.

Правила представления модератора в виде произведения двух модераторов слева и справа от нечеткого неравенства:

$$\frac{\Delta, F_1(q \cdot A \prec'' B)F_2}{\Delta, F_1(q_1 \cdot q_2 \cdot A \prec'' B)F_2}\ (q_1 \cdot q_2 \prec), \qquad \frac{\Delta, F_1(A \prec'' q \cdot B)F_2}{\Delta, F_1(A \prec'' q_1 \cdot q_2 \cdot B)F_2}\ (\prec q_1 \cdot q_2),$$

где $\prec''$ — либо всюду $\prec_+$, либо всюду $\prec_-$.

Правила введения конъюнкции слева и справа от положительного вхождения нечеткого неравенства:

$$\frac{\Delta, F_1(A \prec_+ q_p \cdot B)F_2 \quad \Delta, F_1(A \prec_+ q_p \cdot C)F_2}{\Delta, F_1(A \prec_+ q_p \cdot (B\&C))F_2}\ (\prec_+ q_p\&),$$

$$\frac{\Delta, F_1(q_n \cdot A \prec_+ C)F_2 \quad \Delta, F_1(q_n \cdot B \prec_+ C)F_2}{\Delta, F_1(q_n \cdot (A\&B) \prec_+ C)F_2}\ (q_n\& \prec_+),$$

$$\frac{\Delta, F_1(q_p \cdot A \prec_+ C)F_2, F_1(q_p \cdot B \prec_+ C)F_2}{\Delta, F_1(q_p \cdot (A\&B) \prec_+ C)F_2}\ (q_p\& \prec_+),$$

$$\frac{\Delta, F_1(A \prec_+ q_n \cdot B)F_2, F_1(A \prec_+ q_n \cdot C)F_2}{\Delta, F_1(A \prec_+ q_n \cdot (B\&C))F_2}\ (\prec_+ q_n\&).$$

Правила введения дизъюнкции слева и справа от положительного вхождения нечеткого неравенства:

$$\frac{\Delta, F_1(q_p \cdot A \prec_+ C)F_2 \quad \Delta, F_1(q_p \cdot B \prec_+ C)F_2}{\Delta, F_1(q_p \cdot (A \vee B) \prec_+ C)F_2}\ (q_p\vee \prec_+),$$

$$\frac{\Delta, F_1(A \prec_+ q_n \cdot B)F_2 \quad \Delta, F_1(A \prec_+ q_n \cdot C)F_2}{\Delta, F_1(A \prec_+ q_n \cdot (B \vee C))F_2}\ (\prec_+ q_n\vee),$$

$$\frac{\Delta, F_1(A \prec_+ q_p \cdot B)F_2, F_1(A \prec_+ q_p \cdot C)F_2}{\Delta, F_1(A \prec_+ q_p \cdot (B \vee C))F_2}\ (\prec_+ q_p\vee),$$

$$\frac{\Delta, F_1(q_n \cdot A \prec_+ C)F_2, F_1(q_n \cdot B \prec_+ C)F_2}{\Delta, F_1(q_n \cdot (A \vee B) \prec_+ C)F_2}\ (q_n\vee \prec_+).$$

Правила введения квантора всеобщности слева и справа от положительного вхождения нечеткого неравенства:

$$\frac{\Delta, F_1(q_p \cdot A(t|x) \prec_+ B)F_2, F_1(q_p \cdot \forall x A \prec_+ B)F_2}{\Delta, F_1(q_p \cdot \forall x A \prec_+ B)F_2}\ (q_p\forall \prec_+),$$

$$\frac{\Delta, F_1(A \prec_+ q_p \cdot B(y|x))F_2}{\Delta, F_1(A \prec_+ q_p \cdot \forall x B)F_2}\ (\prec_+ q_p\forall), \quad \frac{\Delta, F_1(q_n \cdot A(y|x) \prec_+ B)F_2}{\Delta, F_1(q_n \cdot \forall x A \prec_+ B)F_2}\ (q_n\forall \prec_+),$$

$$\frac{\Delta, F_1(A \prec_+ q_n \cdot B(t|x))F_2, F_1(A \prec_+ q_n \forall x B)F_2}{\Delta, F_1(A \prec_+ q_n \cdot \forall x B)F_2}\ (\prec_+ q_n\forall).$$

Правила введения квантора существования слева и справа от положительного вхождения нечеткого неравенства:

$$\frac{\Delta, F_1(A \prec_+ q_p \cdot B(t|x))F_2, F_1(A \prec_+ q_p \cdot \exists x B)F_2}{\Delta, F_1(A \prec_+ q_p \cdot \exists x B)F_2}\ (\prec_+ q_p\exists),$$

$$\frac{\Delta, F_1(q_p \cdot A(y|x) \prec_+ B)F_2}{\Delta, F_1(q_p \cdot \exists x A \prec_+ B)F_2}\ (q_p\exists \prec_+), \quad \frac{\Delta, F_1(A \prec_+ q_n \cdot B(y|x))F_2}{\Delta, F_1(A \prec_+ q_n \cdot \exists x B)F_2}\ (\prec_+ q_n\exists),$$

$$\frac{\Delta, F_1(q_n \cdot A(t|x) \prec_+ B)F_2, F_1(q_n \cdot \exists x A \prec_+ B)F_2}{\Delta, F_1(q_n \cdot \exists x A \prec_+ B)F_2}\ (q_n\exists \prec_+).$$

Правила введения конъюнкции слева и справа от отрицательного вхождения нечеткого неравенства:

$$\frac{\begin{array}{l}\Delta, F_1(A \prec_- q_n \cdot B)F_2\\ \Delta, F_1(A \prec_- q_n \cdot C)F_2\end{array}}{\Delta, F_1(A \prec_- q_n \cdot (B\&C))F_2}\ (\prec_- q_n\&),$$

$$\frac{\begin{array}{l}\Delta, F_1(q_p \cdot A \prec_- C)F_2\\ \Delta, F_1(q_p \cdot B \prec_- C)F_2\end{array}}{\Delta, F_1(q_p \cdot (A\&B) \prec_- C)F_2}\ (q_p\& \prec_-),$$

$$\frac{\Delta, F_1(q_n \cdot A \prec_- C)F_2, F_1(q_n \cdot B \prec_- C)F_2}{\Delta, F_1(q_n \cdot (A\&B) \prec_- C)F_2}\ (q_n\& \prec_-),$$

$$\frac{\Delta, F_1(A \prec_- q_p \cdot B)F_2, F_1(A \prec_- q_p \cdot C)F_2}{\Delta, F_1(A \prec_- q_p \cdot (B\&C))F_2}\ (\prec_- q_p\&).$$

Правила введения дизъюнкции слева и справа от отрицательного вхождения нечеткого неравенства:

$$\frac{\begin{array}{l}\Delta, F_1(q_n \cdot A \prec_- C)F_2\\ \Delta, F_1(q_n \cdot B \prec_- C)F_2\end{array}}{\Delta, F_1(q_n \cdot (A \vee B) \prec_- C)F_2}\ (q_n\vee \prec_-),$$

$$\frac{\begin{array}{l}\Delta, F_1(A \prec_- q_p \cdot B)F_2\\ \Delta, F_1(A \prec_- q_p \cdot C)F_2\end{array}}{\Delta, F_1(A \prec_- q_p \cdot (B \vee C))F_2}\ (\prec_- q_p\vee),$$

$$\frac{\Delta, F_1(A \prec_- q_n \cdot B)F_2, F_1(A \prec_- q_n \cdot C)F_2}{\Delta, F_1(A \prec_- q_n \cdot (B \vee C))F_2}\ (\prec_- q_n\vee),$$

$$\frac{\Delta, F_1(q_p \cdot A \prec_- C)F_2, F_1(q_p \cdot B \prec_- C)F_2}{\Delta, F_1(q_p \cdot (A \vee B) \prec_- C)F_2}\ (q_p\vee \prec_-).$$

Правила введения квантора всеобщности слева и справа от отрицательного вхождения нечеткого неравенства:

$$\frac{\Delta, F_1(q_n \cdot A(t|x) \prec_- B)F_2, F_1(q_n \cdot \forall x A \prec_- B)F_2}{\Delta, F_1(q_n \cdot \forall x A \prec_- B)F_2}\ (q_n\forall \prec_-),$$

$$\frac{\Delta, F_1(A \prec_- q_n \cdot B(y|x))F_2}{\Delta, F_1(A \prec_- q_n \cdot \forall x B)F_2}\ (\prec_- q_n\forall), \quad \frac{\Delta, F_1(q_p \cdot A(y|x) \prec_- B)F_2}{\Delta, F_1(q_p \cdot \forall x A \prec_- B)F_2}\ (q_p\forall \prec_-),$$

$$\frac{\Delta, F_1(A \prec_- q_p \cdot B(t|x))F_2, F_1(A \prec_- q_p\forall x B)F_2}{\Delta, F_1(A \prec_- q_p \cdot \forall x B)F_2}\ (\prec_- q_p\forall).$$

Правила введения квантора существования слева и справа от отрицательного вхождения нечеткого неравенства:

$$\frac{\Delta, F_1(A \prec_- q_n \cdot B(t|x))F_2, F_1(A \prec_- q_n \cdot \exists x B)F_2}{\Delta, F_1(A \prec_- q_n \cdot \exists x B)F_2}\ (\prec_- q_n\exists),$$

$$\frac{\Delta, F_1(q_n \cdot A(y|x) \prec_- B)F_2}{\Delta, F_1(q_n \cdot \exists x A \prec_- B)F_2}\ (q_n\exists \prec_-), \quad \frac{\Delta, F_1(A \prec_- q_p \cdot B(y|x))F_2}{\Delta, F_1(A \prec_- q_p \cdot \exists x B)F_2}\ (\prec_- q_p\exists),$$

$$\frac{\Delta, F_1(q_p \cdot A(t|x) \prec_- B)F_2, F_1(q_p \cdot \exists x A \prec_- B)F_2}{\Delta, F_1(q_p \cdot \exists x A \prec_- B)F_2}\ (q_p\exists \prec_-).$$

Также используются правила вывода, полученные из всех упомянутых выше правил, в которые входит $q_p\cdot$, кроме правила $(q_p \prec)$, путем удаления $q_p\cdot$. (Для таких правил новые обозначения не вводятся: если модератор перед формулой отсутствует, то можно считать, что перед этой формулой стоит модератор $1\cdot$.)

Кванторные правила вывода имеют следующие ограничения.

(1) Все упомянутые подстановки ($A(t|x)$, $A(y|x)$, $B(t|x)$, $B(y|x)$) являются свободными.

(2) Для каждого кванторного правила вывода, в котором фигурирует предметная переменная y, эта переменная не входит свободно в заключение правила.

(3) Для каждого кванторного правила вывода предметная переменная x не входит свободно в заключение правила и не входит связанно в подформулу A или B минимальной подформулы заключения, содержащей введенный этим правилом квантор.

Замечание 2.2.1.6. Ограничение (3) на кванторные правила гарантирует получение только чистых секвенций-заключений из чистых секвенций-посылок (в соответствии с определением чистой секвенции 2.2.0.5).

Замечание 2.2.1.7. Каждое из упомянутых правил вывода вводит определенный логический символ в заключение этого правила, кроме разве лишь правил $(q_1 \cdot q_2)$, $(q_p \prec)$, $(q_n \prec)$, $(q_1 \cdot q_2 \prec)$ и $(\prec q_1 \cdot q_2)$. Нам будет удобно говорить, что все упомянутые правила вывода вводят логический символ. При этом будем считать, что правило $(q_1 \cdot q_2)$ вводит модератор $q_1\cdot$, правило $(q_p \prec)$ — $q_p\cdot$, правило $(q_n \prec)$ — $q_n\cdot$, правила $(q_1 \cdot q_2 \prec)$ и $(\prec q_1 \cdot q_2)$ — $q_1\cdot$.

Замечание 2.2.1.8. Исчисление намеренно сформулировано так, что в нем нет правил, изменяющих структуру секвенции без введения логического символа, в частности, в исчислении нет правил, переставляющих формулы в секвенциях. Это улучшает удобочитаемость вывода.

Определение 2.2.1.9. Формула, являющаяся членом секвенции-заключения некоторого правила вывода, называется *главной*, если в эту формулу входит введенный этим правилом логический символ. Формулы, явно указанные в посылках, называются *боковыми*.

Определение 2.2.1.10. Правила вывода, вводящие один из кванторов, называются *кванторными*. Остальные правила вывода называются *пропозициональными*. Правила вывода, в записи которых фигурирует терм t, назовем *минус-правилами*[3]. Остальные кванторные правила назовем *плюс-правилами*.

[3]Ср. [34].

Определение 2.2.1.11. Предметную переменную y каждого плюс-правила называют *собственной* переменной этого правила.

Обозначение 2.2.1.12. Если из правил вывода исчисления LqS оставить только пропозициональные, то получится исчисление, обозначаемое нами LqS_{propos}.

Замечание 2.2.1.13. Вывод зачастую полезно дополнять указанием на то, из каких секвенций и применением какого правила получена секвенция, или указанием на то, что секвенция является аксиомой.

Еще более наглядно и удобно для анализа представлять вывод в виде дерева, которое строится согласно следующим определениям.

Определение 2.2.1.14. *Деревом поиска вывода* секвенции S называется дерево, корень которого является (или помечен) секвенцией S, и если секвенция S является заключением применения правила вывода (R) к секвенциям $S_1, \ldots, S_n$, то узел S имеет все секвенции $S_1, \ldots, S_n$ своими непосредственными потомками, а каждый из узлов S_i является деревом поиска вывода секвенции S_i. Считается, что корень такого дерева располагается снизу, и дерево растет вверх. (Каждый узел дерева можно дополнительно помечать необходимой для анализа информацией, например, правилом вывода, заключением которого является секвенция, представляемая этим узлом.)

Определение 2.2.1.15. *Деревом вывода* секвенции называется дерево поиска вывода этой секвенции, если все листья этого дерева являются аксиомами.

Имея в виду представление вывода в виде дерева, будем говорить о ветвях дерева и о том, что одно вхождение секвенции находится выше другого, если первое выше второго в одной и той же ветви.

Замечание 2.2.1.16. Каждое правило вывода таково, что его посылки содержат лишь формулы, совпадающие с формулами заключения или полученные путем расщепления (т.е. исключения логического символа) одной

из формул заключения. Более точно, назовем «подформулой» формулы A саму эту формулу и всевозможные боковые формулы, которые могут находиться в посылке правила вывода с главной формулой A. (Можно дать определение «подформулы», не упоминающее правила вывода. Мы не приводим такое очевидное, но довольно громоздкое определение.) Таким образом, в выводе (и дереве поиска вывода) секвенции встречаются лишь «подформулы» этой секвенции. В связи с этим будем говорить, что предложенное секвенциальное исчисление обладает *свойством подформульности*. Это свойство позволяет легко подбирать посылки по известному заключению для всех правил вывода, кроме разве лишь минус-правил.

Определение 2.2.1.17. Если секвенция S является заключением применения правила (R) к секвенциям-посылкам $S_1, \ldots, S_n$, то говорят также, что произведено *контрприменение* правила (R) к секвенции S и в результате этого контрприменения получены секвенции $S_1, \ldots, S_n$.

Замечание 2.2.1.18. Одно из преимуществ данного нами определения чистой секвенции по сравнению с определением чистоты из [24] (см. определение 2.2.0.5 на с. 37 и следующее за ним замечание) состоит в том, что в результате контрприменения любого правила вывода к секвенции получаются посылки, являющиеся чистыми секвенциями. Если бы использовалось определение чистоты из [24], то это было бы не так. Действительно, главная формула минус-правила является как членом заключения, так и членом посылки, и членом посылки является также боковая формула, полученная из главной подстановкой терма. Поэтому в посылку минус-правила может дважды входить одна и та же подформула главной формулы, а в эту подформулу может входить кванторная приставка.

2.2.2. Аксиомы

Определение 2.2.2.1. *Каноническая цепь неравенств (КЦН)* определяется следующим образом. Формулы вида P и $q \cdot P$ (где $q\cdot$ — модератор, P — атомарная формула) являются КЦН. Если I и J — КЦН, то $(I \prec J)$

является КЦН.

Определение 2.2.2.2. Пусть дана секвенция S. Удалим из S все члены, которые не являются КЦН; тогда полученная секвенция называется *базовой подсеквенцией* секвенции S.

Определение 2.2.2.3. Секвенция называется *аксиомой*, если базовая подсеквенция этой секвенции общезначима.

Любой секвенции с непустой базовой подсеквенцией сопоставим систему линейных неравенств, которая строится из базовой подсеквенции следующим образом.

Все вхождения одной и той же пропозициональной переменной (одного и того же предикатного символа с одним и тем же списком аргументов) заменяются одной и той же действительнозначной переменной, причем разным пропозициональным переменным (разным предикатным символам со списками аргументов) соответствуют разные переменные. Каждое выражение вида $(x \prec y)$, где x и y — действительнозначные переменные, заменяется на $(y - x)$. Каждый модератор вида $q\cdot$ трактуется как умножение на рациональное число q. Пусть выражение $\tilde{A}$ соответствует члену A базовой подсеквенции при только что описанном преобразовании. Тогда для каждого члена A базовой подсеквенции числовое неравенство $\tilde{A} < 0$ включается в систему. Наконец, пусть действительнозначная переменная соответствует некоторой пропозициональной переменной (предикатному символу со списком аргументов); тогда для каждой такой переменной x, неравенства $a \leq x$, $x \leq b$, где $[a, b]$ — отрезок истинностных значений этой пропозициональной переменной (предикатного символа), включаются в систему.

Теорема 2.2.2.4. *Непустая базовая подсеквенция какой угодно секвенции общезначима тогда и только тогда, когда система линейных неравенств, построенная вышеуказанным образом, несовместна.*

Доказательство. Пусть базовая подсеквенция имеет вид

$$A_1, \ \ldots, \ A_n,$$

где каждая из формул A_i является КЦН. Тогда общезначимость базовой подсеквенции эквивалентна тому, что во всякой интерпретации при любой оценке

$$[A_1] \geq 0 \text{ или } \ldots \text{ или } [A_n] \geq 0.$$

А то, что базовая подсеквенция не общезначима, эквивалентно тому, что найдутся интерпретация и оценка, при которых

$$[A_1] < 0 \text{ и } \ldots \text{ и } [A_n] < 0; \tag{2.1}$$

причем по построению системы каждое из неравенств системы (кроме тех неравенств системы, которые отражают ограничения, связанные с отрезками истинностных значений) получается из взаимно-однозначно соответствующего ему неравенства из (2.1) путем замены атомарных формул на числовые переменные.

Необходимость. Если система совместна, то сопоставление компонент ее вектора-решения соответствующим атомарным формулам базовой подсеквенции определяет интерпретацию и оценку, при которых выполняется (2.1), и потому базовая подсеквенция не является общезначимой. Таким образом, общезначимость базовой подсеквенции влечет несовместность системы.

Достаточность. Если базовая секвенция не является общезначимой, то присваивание значений атомарных формул при интерпретации и оценке, при которых выполняется (2.1), соответствующим переменным системы задает решение системы неравенств, и потому система совместна. Таким образом, несовместность системы влечет общезначимость базовой подсеквенции. □

Замечание 2.2.2.5. Система линейных неравенств с рациональными коэффициентами совместна или несовместна вне зависимости от того, считать ли все ее переменные действительнозначными или рациональнозначными.

Итак, для проверки того, является ли секвенция аксиомой, требуется проверить несовместность соответствующей системы строгих и нестро-

гих линейных неравенств с рациональными коэффициентами и рациональнозначными переменными.

Замечание 2.2.2.6. Эта задача отличается от классической задачи линейного программирования по проверке совместности системы линейных неравенств (см., например, [46]) тем, что в системе могут быть и строгие, и нестрогие неравенства, тогда как в классической задаче линейного программирования допустимы только нестрогие неравенства. Однако для решения данной задачи может быть применена модификация некоторого алгоритма линейного программирования. Существуют полиномиальные алгоритмы решения данной задачи, основанные на алгоритмах Хачияна и Кармаркара (см. [28, 29]). Но сильно полиномиальный алгоритм для решения этой задачи не известен в настоящее время (см. раздел 1.1.4).

2.3. Некоторые свойства исчисления

2.3.1. Семантическое обоснование исчисления

Лемма 2.3.1.1. *В исчислении LqS*

- *все аксиомы общезначимы;*
- *заключение каждого правила вывода общезначимо, если и только если каждая посылка этого правила общезначима.*

Доказательство. Аксиома содержит базовую подсеквенцию, которая общезначима по определению, и поэтому очевидно, что аксиома является общезначимой.

Дальнейшее доказательство представляет собой проверку общезначимости заключения каждого правила вывода при условии общезначимости посылки и проверку общезначимости каждой посылки каждого правила при условии общезначимости заключения. Такие проверки осуществляется однотипным образом. Поэтому рассмотрим лишь несколько правил.

1a. Пусть посылки следующего правила общезначимы:

$$\frac{\Delta, F_1(A \prec_+ q_p \cdot B)F_2 \quad \Delta, F_1(A \prec_+ q_p \cdot C)F_2}{\Delta, F_1(A \prec_+ q_p \cdot (B\&C))F_2}\ (\prec_+ q_p\&).$$

Тогда во всякой интерпретации при любой оценке

- верно $[\Phi(\Delta)] \geq 0$ или $[F_1(A \prec_+ q_p \cdot B)F_2] \geq 0$, и
- верно $[\Phi(\Delta)] \geq 0$ или $[F_1(A \prec_+ q_p \cdot C)F_2] \geq 0$.

Если $[\Phi(\Delta)] \geq 0$, то $[\Phi(\Delta, F_1(A \prec_+ q_p \cdot (B\&C))F_2)] \geq 0$, что и требуется.

Иначе должно выполняться $[F_1(A \prec_+ q_p \cdot B)F_2] \geq 0$ и $[F_1(A \prec_+ q_p \cdot C)F_2] \geq 0$.

Учитывая то, что выделенное вхождение нечеткого неравенства положительно, формулы B и C находятся справа от него, и перед этими формулами стоит модератор с неотрицательным числом, последние два неравенства можно переписать в виде $d + q_p \cdot [B] \geq 0$ и $d + q_p \cdot [C] \geq 0$ соответственно, где d — действительное число. Отсюда получаем $d + q_p \cdot \min([B], [C]) \geq 0$, а это есть другая запись выражения $[F_1(A \prec_+ q_p \cdot (B\&C))F_2] \geq 0$. Таким образом, и в этом случае $[\Phi(\Delta, F_1(A \prec_+ q_p \cdot (B\&C))F_2)] \geq 0$.

1b. Пусть заключение следующего правила вывода общезначимо:

$$\frac{\Delta, F_1(A \prec_+ q_p \cdot B)F_2 \quad \Delta, F_1(A \prec_+ q_p \cdot C)F_2}{\Delta, F_1(A \prec_+ q_p \cdot (B\&C))F_2}\ (\prec_+ q_p\&).$$

Тогда во всякой интерпретации при любой оценке верно $[\Phi(\Delta)] \geq 0$ или $[F_1(A \prec_+ q_p \cdot (B\&C))F_2] \geq 0$.

Если $[\Phi(\Delta)] \geq 0$, то $[\Phi(\Delta, F_1(A \prec_+ q_p \cdot B)F_2)] \geq 0$.

Иначе должно выполняться $[F_1(A \prec_+ q_p \cdot (B\&C))F_2] \geq 0$. Как и в п. 1a, последнее неравенство можно переписать в виде $d + q_p \cdot \min([B], [C]) \geq 0$, где d — действительное число. Отсюда $d + q_p \cdot [B] \geq 0$ (соответственно, $d + q_p \cdot [C] \geq 0$), что является другой записью выражения $[F_1(A \prec_+ q_p \cdot B)F_2] \geq 0$ (соответственно, $[F_1(A \prec_+ q_p \cdot C)F_2] \geq 0$). Таким образом, получаем $[\Phi(\Delta, F_1(A \prec_+ q_p \cdot B)F_2)] \geq 0$ (соответственно, $[\Phi(\Delta, F_1(A \prec_+ q_p \cdot C)F_2)] \geq 0$).

2a. Пусть посылка следующего правила общезначима:

$$\frac{\Delta, F_1(A \prec_- q_n \cdot B(t|x))F_2, F_1(A \prec_- q_n \cdot \exists x B)F_2}{\Delta, F_1(A \prec_- q_n \cdot \exists x B)F_2} \ (\prec_- q_n\exists).$$

Тогда во всякой интерпретации с носителем D при любой оценке верно $[\Phi(\Delta)] \geq 0$ или $[F_1(A \prec_- q_n \cdot B(t|x))F_2] \geq 0$ или $[F_1(A \prec_- q_n \cdot \exists x B)F_2] \geq 0$.

Если $[\Phi(\Delta)] \geq 0$, то $[\Phi(\Delta, F_1(A \prec_- q_n \cdot \exists x B)F_2)] \geq 0$.

Иначе должно выполняться $[F_1(A \prec_- q_n \cdot B(t|x))F_2] \geq 0$ или $[F_1(A \prec_- q_n \cdot \exists x B)F_2] \geq 0$.

Учитывая то, что выделенное вхождение нечеткого неравенства отрицательно, формулы $B(t|x)$ и $\exists x B$ находятся справа от него, и перед этими формулами стоит модератор с отрицательным числом, последние два неравенства можно переписать в виде $d - q_n \cdot [B(t|x)] \geq 0$ и $d - q_n \cdot [\exists x B] \geq 0$ соответственно, где d — действительное число. Так как $[\exists x B] \geq [B(t|x)]$, то $d - q_n \cdot [B(t|x)] \geq 0$ влечет $d - q_n \cdot [\exists x B] \geq 0$. Поэтому $[F_1(A \prec_- q_n \cdot \exists x B)F_2] \geq 0$, следовательно, $[\Phi(\Delta, F_1(A \prec_- q_n \cdot \exists x B)F_2)] \geq 0$.

2b. Пусть заключение следующего правила вывода общезначимо:

$$\frac{\Delta, F_1(A \prec_- q_n \cdot B(t|x))F_2, F_1(A \prec_- q_n \cdot \exists x B)F_2}{\Delta, F_1(A \prec_- q_n \cdot \exists x B)F_2} \ (\prec_- q_n\exists).$$

Общезначимость посылки этого правила следует из общезначимости заключения, поскольку все члены заключения являются членами посылки.

3a. Пусть посылка следующего правила общезначима:

$$\frac{\Delta, q_n \cdot A(y|x)}{\Delta, q_n \cdot \exists x A} \ (q_n\exists).$$

Тогда во всякой интерпретации I с носителем D и при любой оценке v верно $[\Phi(\Delta)]_{I,v} \geq 0$ или $[q_n \cdot A(y|x)]_{I,v} \geq 0$.

Если $[\Phi(\Delta)]_{I,v} \geq 0$, то $[\Phi(\Delta, q_n \cdot \exists x A)]_{I,v} \geq 0$.

Иначе должно выполняться $[q_n \cdot A(y|x)]_{I,v} \geq 0$. Последнее неравенство эквивалентно $[A(y|x)]_{I,v} \leq 0$.

Из ограничения на кванторное правило следует, что y не входит свободно в Δ. Поэтому $[A(y|x)]_{I,v'} \leq 0$ для любой оценки v', возможно отли-

чающейся от оценки v только образом переменной y. Отсюда получаем $\sup\{[A(y|x)]_{I,v(b|y)}|b \in D\} \leq 0$. Следовательно, поскольку y не входит в A свободно (в соответствии с ограничением на кванторное правило), имеем $[\exists x A]_{I,v} \leq 0$. Таким образом, $[\Phi(\Delta, q_n \cdot \exists x A)]_{I,v} \geq 0$.

3b. Пусть заключение следующего правила вывода общезначимо:
$\dfrac{\Delta, q_n \cdot A(y|x)}{\Delta, q_n \cdot \exists x A}$ $(q_n\exists)$.

Пусть посылка правила $(q_n\exists^-)$ общезначима. Тогда во всякой интерпретации I с носителем D и при любой оценке v верно $[\Phi(\Delta)]_{I,v} \geq 0$ или $[q_n \cdot \exists x A]_{I,v} \geq 0$.

Если $[\Phi(\Delta)]_{I,v} \geq 0$, то $[\Phi(\Delta, q_n \cdot A(y|x))]_{I,v} \geq 0$.

Иначе должно выполняться $[q_n \cdot \exists x A]_{I,v} \geq 0$. Последнее неравенство эквивалентно $[\exists x A]_{I,v} \leq 0$, что можно переписать в виде $\sup\{[A(y|x)]_{I,v(b|y)}|b \in D\} \leq 0$, поскольку y не входит в A свободно в соответствии с ограничением на кванторное правило. Поэтому $[A(y|x)]_{I,v} \leq 0$, следовательно, $[\Phi(\Delta, q_n \cdot A(y|x))]_{I,v} \geq 0$. □

Определение 2.3.1.2. Исчисление для логики Lq называется *семантически обоснованным*, если все выводимые в этом исчислении формулы общезначимы.

Теорема 2.3.1.3 (Семантическое обоснование исчисления LqS)**.** *Если секвенция выводима в исчислении LqS, то она общезначима.*

Доказательство. Воспользуемся методом возвратной математической индукции по длине вывода секвенции S (т.е. по числу секвенций в выводе секвенции S).

База индукции. Если S — аксиома, то она общезначима по лемме 2.3.1.1.

Предположим, что последняя секвенция в выводах длиной, не большей n, общезначима.

Пусть дан вывод длиной $n+1$ с последней секвенцией S. Секвенция S получена из одной или двух секвенций-посылок по некоторому правилу

вывода, причем длина вывода каждой из секвенций-посылок не превосходит n. Поэтому к этим секвенциям применимо индукционное предположение, следовательно, они общезначимы. Тогда по лемме 2.3.1.1 секвенция S общезначима. □

Лемма 2.3.1.4. *Пусть T — дерево поиска вывода секвенции S в исчислении LqS, причем для каждой листовой секвенции S_{leaf} этого дерева ее базовая подсеквенция совпадает с самой секвенцией S_{leaf}. Тогда секвенция S выводима, если и только если каждая листовая секвенция дерева T является аксиомой.*

Доказательство. Достаточность очевидна. Докажем необходимость.

Так как секвенция S выводима, то по теореме 2.3.1.3 эта секвенция общезначима. Следовательно, по лемме 2.3.1.1 общезначима каждая секвенция дерева поиска вывода T, в том числе и каждая листовая секвенция. Поскольку каждая листовая секвенция общезначима и совпадает со своей базовой подсеквенцией, то по определению аксиомы каждая листовая секвенция является аксиомой. □

Следующая лемма будет полезной для установления невыводимости секвенций из весьма широкого класса секвенций.

Лемма 2.3.1.5. *Пусть дано дерево поиска вывода секвенции S в исчислении LqS. Тогда, если найдется листовая секвенция S_{leaf} этого дерева, базовая подсеквенция которой совпадает с самой секвенцией S_{leaf}, и S_{leaf} не является аксиомой, то секвенция S невыводима.*

Доказательство. Секвенция S_{leaf} не является аксиомой и совпадает со своей базовой подсеквенцией, следовательно, по определению аксиомы секвенция S_{leaf} необщезначима. Тогда по лемме 2.3.1.1 секвенция S тоже необщезначима. Поэтому по теореме 2.3.1.3 секвенция S невыводима. □

2.3.2. Непротиворечивость исчисления

Определение 2.3.2.1. Исчисление называется *непротиворечивым*, если не все объекты этого исчисления выводимы.

Предложение 2.3.2.2 (Непротиворечивость исчисления LqS)**.** *Существует секвенция, невыводимая в исчислении LqS.*

Доказательство. Невыводимыми, например, являются секвенции, состоящие из отрицательного рационального числа, и пустая секвенция. В самом деле, аксиомы непусты, в заключении каждого правило вывода имеется член, содержащий логический символ. Таким образом, указанные секвенции не могут быть получены ни по какому правилу вывода и не являются аксиомами, следовательно, такие секвенции невыводимы. □

2.3.3. Связь с исчислением для классической двузначной логики

Рассмотрим классическую двузначную логику первого порядка с логическими символами & (конъюнкция), $\vee$ (дизъюнкция), $\neg$ (отрицание), $\forall$ (квантор всеобщности), $\exists$ (квантор существования) и истинностными константами T (истина), F (ложь). Формулы этой двузначной логики строятся следующим образом.

Термами являются предметные переменные и предметные константы.

Атомарными формулами являются истинностные константы, пропозициональные переменные и выражения вида $P(t_1, \ldots, t_n)$, где P — n-местный предикатный символ, n — положительное целое число, $t_1, \ldots, t_n$ — термы.

Атомарные формулы являются формулами двузначной логики. Если A и B — формулы двузначной логики, x — предметная переменная, то $\neg A$, $(A\&B)$, $(A \vee B)$, $\forall x A$ и $\exists x A$ являются формулами двузначной логики.

Безантецедентное секвенциальное исчисление LS для этой логики формулируется следующим образом. Понятие секвенции определяется так же

как в разделе 2.2, только под формулой понимается формула двузначной логики. Используются такие же обозначения и применяются те же ограничения, что и при формулировке в разделе 2.2.1 правил вывода исчисления LqS. Правила вывода исчисления LS имеют вид:

$$\frac{\Delta, A}{\Delta, \neg\neg A}\ (\neg\neg), \qquad \frac{\Delta, A \quad \Delta, B}{\Delta, (A\&B)}\ (\&), \qquad \frac{\Delta, \neg A \quad \Delta, \neg B}{\Delta, \neg(A \vee B)}\ (\neg\vee),$$

$$\frac{\Delta, \neg A, \neg B}{\Delta, \neg(A\&B)}\ (\neg\&), \qquad \frac{\Delta, A, B}{\Delta, (A \vee B)}\ (\vee), \qquad \frac{\Delta, A(t|x), \exists x A}{\Delta, \exists x A}\ (\exists),$$

$$\frac{\Delta, A(y|x)}{\Delta, \forall x A}\ (\forall), \qquad \frac{\Delta, \neg A(y|x)}{\Delta, \neg\exists x A}\ (\neg\exists), \qquad \frac{\Delta, \neg A(t|x), \neg\forall x A}{\Delta, \neg\forall x A}\ (\neg\forall).$$

Видно, что эти правила вывода соответствуют правилам из первого блока правил исчисления LqS: эти правила получаются из указанных правил исчисления LqS с помощью замены модераторов $q_1\cdot$, $q_2\cdot$, $q_n\cdot$ на отрицание и удаления модераторов $q\cdot$, $q_p\cdot$.

Аксиомами исчисления LS являются секвенции вида:

(a) Δ, T;

(b) $\Delta, \neg F$;

(c) $\Delta, P, \neg P$, где P — атомарная формула.

Зададим отображение χ, преобразующее формулы двузначной логики в формулы логики Lq. Пусть A — формула двузначной логики. $\chi(A)$ является формулой логики Lq, которая строится из A таким образом: каждое вхождение T заменяется на 1, каждое вхождение F заменяется на -1, каждое вхождение $\neg$ заменяется на модератор $-1\cdot$; к каждому предикатному символу (пропозициональной переменной) дописывается отрезок истинностных значений $[-1, 1]$.

Естественным образом продолжим отображение χ на секвенции: если S — секвенция с членами $A_1, \ldots, A_n$, то $\chi(S)$ есть секвенция с членами $\chi(A_1), \ldots, \chi(A_n)$.

Ясно, что χ — это инъективное отображение множества секвенций исчисления LS в множество секвенций исчисления LqS.

Определение 2.3.3.1. Пусть даны два исчисления C и C', языками которых являются L и L' соответственно. Будем говорить, что исчисление C *вкладывается* в исчисление C', если существует инъективное отображение $\psi: \ L \to L'$, и для любого $w \in L \ \ w$ выводим в C тогда и только тогда, когда $\psi(w)$ выводим в C'.

Теорема 2.3.3.2. *Исчисление LS для классической двузначной логики вкладывается в исчисление LqS.*

Доказательство этой теоремы сводится к следующей лемме. □

Лемма 2.3.3.3. *Пусть S — секвенция исчисления LS. Секвенция S выводима в исчислении LS тогда и только тогда, когда секвенция $\chi(S)$ выводима в исчислении LqS.*

Доказательство. Покажем, что вывод секвенции S можно перестроить в вывод секвенции $\chi(S)$ и наоборот.

Необходимость. Пусть дан вывод секвенции S в исчислении LS. Каждую секвенцию S_0 этого вывода заменим на $\chi(S_0)$. Получим список секвенций исчисления LqS с последней секвенцией $\chi(S)$. Этот список является выводом, поскольку очевидно, что каждое применение правила исчисления LS, переходит в применение соответствующего правила исчисления LqS, и любая аксиома исчисления LS переходит в аксиому исчисления LqS.

Достаточность. Пусть дан вывод секвенции $\chi(S)$ в исчислении LqS. Каждую секвенцию S_1 этого вывода заменим на $\chi^{-1}(S_1)$ — результат обратного к χ преобразования. Получим список секвенций исчисления LS с последней секвенцией S. При этом, очевидно, каждое применение правила исчисления LqS в выводе секвенции $\chi(S)$ переходит в применение правила исчисления LS. Чтобы утверждать, что полученный список является выводом, осталось установить, что любая аксиома исчисления LqS, которая

может встретиться в выводе секвенции $\chi(S)$, переходит в аксиому исчисления LS.

Нетрудно видеть, что любая аксиома исчисления LqS, которая может встретиться в выводе секвенции $\chi(S)$, имеет один из следующих видов:

(a) $\Delta, 1$;

(b) $\Delta, -1 \cdot -1$;

(c) $\Delta, P, -1 \cdot P$, где P — атомарная формула логики Lq.

Такие аксиомы переходят в аксиомы исчисления LS. □

2.3.4. Вопросы полноты и разрешимости исчисления

Определение 2.3.4.1. Исчисление для логики Lq назовем *полным*, если каждая чистая общезначимая формула логики Lq выводима в этом исчислении.

Определение 2.3.4.2. Исчисление называется *разрешимым*, если существует алгоритм, выясняющий по любому заданному объекту этого исчисления, является он выводимым или нет.

Также нам потребуются следующие определения.

Определение 2.3.4.3. Пусть A — формула классической двузначной логики первого порядка (или формула логики Lq). *Лексемой* формулы A будем называть последовательность неделимых символов, представляющих одну из следующих составных частей формулы: предметная переменная, предметная константа, истинностная константа, предикатный символ, логический символ, открывающая круглая скобка '(', закрывающая круглая скобка ')', запятая ','.

Определение 2.3.4.4. Формулы A и B формулы классической двузначной логики первого порядка. (соответственно, логики Lq) называются *конгруэнтными*, если формулы A и B состоят из одинакового числа k лексем, и для каждого $i = 1, \ldots, k$

- если i-ая лексема формулы A не является вхождением предметной переменной, то эта же лексема является i-ой лексемой формулы B;
- если i-ая лексема формулы A является свободным вхождением предметной переменной, то i-ая лексема формулы B является свободным вхождением той же предметной переменной;
- если i-ая лексема формулы A является вхождением предметной переменной, связанным вхождением квантора, представленным j-ой лексемой, то i-ая лексема формулы B является вхождением какой угодно предметной переменной, связанным вхождением квантора, представленным j-ой лексемой формулы B.

(Т.е., говоря менее точно, конгруэнтные формулы отличаются лишь переименованием связанных переменных, причем в результате этого переименования на местах свободных вхождений предметных переменных не появляются связанные вхождения.)

Определение 2.3.4.5. Два секвенциальных исчисления G' и G'' для классической двузначной логики первого порядка (соответственно, логики Lq) назовем *равнообъемными*, если для любой выводимой в G' формулы A существует формула, конгруэнтная формуле A и выводимая в G'', и для любой выводимой в G'' формулы B существует формула, конгруэнтная формуле B и выводимая в G'.

Замечание 2.3.4.6. Для любой формулы A логики Lq найдется конгруэнтная ей чистая формула A_{pure} логики Lq. Такая чистая формула A_{pure} может быть построена путем переименования связанных переменных формулы A на попарно различные новые переменные. Очевидно, во всякой интерпретации при любой оценке конгруэнтные формулы принимают истинностные значения, которые совпадают друг с другом. В частности, конгруэнтные формулы общезначимы или нет одновременно. Поэтому ясно, что если множество общезначимых формул логики Lq неперечислимо, то и множество чистых общезначимых формул логики Lq неперечислимо.

Учитывая теорему 2.1.3.3 и предыдущее замечание, с помощью тех же рассуждений, что использовались для бесконечнозначной предикатной логики Лукасевича (см. теорему 1.1.3.5), получаем, что имеет место

Теорема 2.3.4.7. *Для логики Lq не существует полное, семантически обоснованное и удовлетворяющее требованию (DAR)*[4] *исчисление.*

Из предыдущей теоремы 2.3.4.7 немедленно получаем

Следствие 2.3.4.8. *Исчисление LqS не является полным.*

Можно показать, что представленное в разделе 2.3.3 безантецедентное секвенциальное исчисление для классической двузначной логики равнообъемно секвенциальным исчислениям для этой логики, представленным, например, в [22, 24]. Поскольку представленные в [22, 24] исчисления для классической двузначной логики не являются разрешимыми (см. там же), то в силу теоремы 2.3.3.2 имеет место

Теорема 2.3.4.9. *Исчисление LqS не является разрешимым.*

Лемма 2.3.4.10. *Пусть S — секвенция исчисления LqS. Тогда S совпадает со своей базовой подсеквенцией S_b, если и только если S не является заключением применения никакого правила вывода исчисления LqS.*

Доказательство.

Необходимость. Если секвенция S является заключением применения некоторого правила вывода, то главная формула ϕ секвенции S содержит логический символ, введенный этим правилом. Просмотрев все правила вывода, убеждаемся, что их главные формулы не могут являться КЦН. Поэтому ϕ не является членом S_b.

Таким образом, если S совпадает со своей базовой подсеквенцией, то S не является заключением применения никакого правила вывода.

Достаточность. Если секвенция S не совпадает со своей базовой подсеквенцией S_b, то выберем член ψ секвенции S, не являющийся членом S_b.

[4] См. с. 11.

Формула ψ не является КЦН, поэтому в дереве формулы ψ (см. определение 2.1.1.7 на с. 32 и замечание 2.2.1.4 на с. 39) найдется узел N такой, что:

- N помечен логическим символом (точнее, логической связкой или кванторной приставкой, частью которой является логический символ — квантор), отличным от $\prec$,
- на пути от корня к N все узлы помечены $\prec$, и
- если N помечен модератором, то непосредственный потомок узла N не помечен атомарной формулой.

(Если бы такой узел N был бы найден в дереве формулы ψ, то легко видеть, что ψ являлась бы КЦН.) Рассмотрев все логические символы, которыми может быть помечен узел N, убеждаемся, что найдется правило, вводящее этот логический символ. Следовательно, секвенция S является заключением применения этого правила.

Таким образом, если S не является заключением применения никакого правила вывода, то S совпадает со своей базовой подсеквенцией. □

Теорема 2.3.4.11. *Исчисление LqS_{propos} полно для пропозиционального фрагмента логики Lq.*

Доказательство. Пусть дана общезначимая пропозициональная формула A логики Lq. Покажем, что A выводима.

Будем строить ее дерево поиска вывода снизу вверх, осуществляя контрприменения правил вывода (в каком угодно порядке) до тех пор, пока не получим в качестве листьев этого дерева секвенции, не содержащие логических связок, которые можно ввести по какому-либо правилу вывода. Ясно, что такое дерево будет получено за конечное число контрприменений правил, и по лемме 2.3.4.10 каждая из полученных листовых секвенций совпадает со своей базовой подсеквенцией.

По лемме 2.3.1.1 формула A общезначима, если и только если каждая листовая секвенция общезначима. Формула A общезначима по условию,

следовательно, каждая листовая секвенция общезначима. Но каждая листовая секвенция совпадает со своей базовой подсеквенцией и потому является аксиомой. Таким образом, построенное дерево поиска вывода является деревом вывода, значит, формула A выводима. □

Теорема 2.3.4.12. *Исчисление LqS_{propos} разрешимо.*

Доказательство. Одним из алгоритмов, выясняющих по заданной формуле A, является она выводимой или нет, будет следующий алгоритм DP. (Этот алгоритм был частично описан в доказательстве предыдущей теоремы; приведем более полное описание.)

Алгоритм DP состоит из двух этапов, приведенных ниже.

1) Строится дерево поиска вывода формулы A снизу вверх. Для этого осуществляются контрприменения правил вывода (в каком угодно порядке) до тех пор, пока все листья дерева не будут секвенциями, не содержащими логических связок, которые можно ввести по какому-либо правилу вывода. (Ясно, что такое дерево будет получено за конечное число контрприменений правил и по лемме 2.3.4.10 каждая из полученных листовых секвенций совпадает со своей базовой подсеквенцией.)

2) С помощью алгоритма, основанного на некотором алгоритме линейного программирования (см. замечание 2.2.2.6 в конце раздела 2.2.2), проверяется, все ли листья являются аксиомами. Если все листья являются аксиомами, то выдается ответ «выводима», иначе — «невыводима».

Чтобы завершить доказательство, нужно показать, что описанный алгоритм DP всегда выдает правильный ответ.

По лемме 2.3.1.4 A выводима тогда и только тогда, когда все листья построенного дерева поиска вывода являются аксиомами. Поэтому алгоритм DP выдаст правильный ответ «выводима», если A выводима, и правильный ответ «невыводима», если A невыводима. □

2.3.5. Минус-нормализация вывода

Рассмотрим поиск вывода секвенции S снизу вверх в исчислении LqS.

Вспомним (см. также раздел 1.1.3), что при таком поиске вывода заданной секвенции S действуют следующим образом. Находят секвенции, из которых секвенция S получается по одному из правил вывода, и для каждой найденной секвенции, если она не является аксиомой, находят секвенции, из которых она получается по одному из правил вывода и т.д.

Привлекая понятие дерева поиска вывода, этот способ можно описать так.

Изначально дерево поиска вывода секвенции S имеет лишь один узел — саму секвенцию S. Затем до тех пор, пока все листья дерева поиска вывода не будут являться аксиомами, выполняются следующие действия. Выбираются листовая секвенция S_{leaf}, ее логический символ, правило вывода (R), вводящее этот логический символ, и секвенции $S_1, \ldots, S_n$, из которых секвенция S_{leaf} получается по выбранному правилу (R). После этого выбора к узлу дерева, представляющего секвенцию S_{leaf}, добавляются непосредственные потомки — секвенции $S_1, \ldots, S_n$.

Таким образом, на очередном шаге поиска осуществляется контрприменение правила (R) к секвенции S_{leaf} и подбор секвенций-посылок $S_1, \ldots, S_n$ для этого контрприменения.

Используя поиск вывода снизу вверх, попытаемся найти вывод секвенции $P(z) \prec \exists x P(x)$, где P — одноместный предикатный символ. Ниже приведено одно из возможных деревьев поиска вывода этой секвенции. (Справа от каждой секвенции указано правило, заключением применения которого является эта секвенция.)

$$\begin{array}{l} P(z) \prec P(a),\ P(z) \prec \exists x P(x) \quad (\prec_+ q_p\exists) \\ \qquad\qquad\qquad\quad | \\ \qquad P(z) \prec \exists x P(x) \quad (\prec_+ q_p\exists) \end{array}$$

Вышеприведенное дерево поиска вывода не является деревом вывода, но если удачно подобрать терм, подставляемый при контрприменении минус-правила $(\prec_+ q_p\exists)$ вместо связанной квантором переменной x,

то можно получить дерево вывода секвенции $P(z) \prec \exists x P(x)$, приведенное ниже.

$$P(z) \prec P(a),\ P(z) \prec P(z),\ P(z) \prec \exists x P(x) \quad \text{(аксиома)}$$
$$\Big|$$
$$P(z) \prec P(a),\ P(z) \prec \exists x P(x) \quad (\prec_+ q_p\exists)$$
$$\Big|$$
$$P(z) \prec \exists x P(x) \quad (\prec_+ q_p\exists)$$

Заметим, что на этот раз подставляемый терм выбран из числа тех термов, которые входят в заключение минус-правила.

Проблема удачного подбора секвенций-посылок при контрприменении правил вывода исчисления LqS возникает лишь с минус-правилами (и эта проблема сводится к удачному подбору термов). При контрприменении остальных правил вывода исчисления LqS, вводящих какой-либо выбранный логический символ, секвенции-посылки подбираются детерминированным образом. Точнее, для пропозициональных правил вывода секвенции-посылки определяются однозначно, а для плюс-правил, хотя и можно выбирать собственную переменную, достаточно выбрать любую, удовлетворяющую ограничениям на кванторные правила.

Итак, существенной проблемой при поиске вывода снизу вверх в исчислении LqS является нахождение термов для подстановки при контрприменениях минус-правил. Таких термов для подстановки бесконечно много.

Для классической двузначной логики первого порядка известны секвенциальные исчисления (см., [35, 21], а также [34, 56, 55]), в которых устранен бесконечный перебор термов для подстановки при контрприменениях минус-правил соответствующих секвенциальных исчислений. Однако упомянутые работы (и все известные нам другие работы) не содержат доказательств равнообъемности предложенных секвенциальных исчислений и какого-либо из традиционных секвенциальных исчислений для классической двузначной логики (например, секвенциальных исчислений, сформулированных в [22]).

В [34] перестройка выводов, приводящая к устранению бесконечного перебора термов для подстановки при контрприменениях минус-правил се-

квенциального исчисления для классической двузначной логики, названа *минус-нормализацией*. Будем применять этот же термин и производные от него по отношению к секвенциальным исчислениям для логики Lq.

Сформулируем исчисление $LqSn$, обладающее свойством минус-нормальности и равнообъемное[5] исчислению LqS.

Исчисление $LqSn$ получается из исчисления LqS лишь добавлением к ограничениям (1)–(3) на кванторные правила (см. с. 43) следующего ограничения (4): для каждого минус-правила вывода

- терм t может быть только одним из термов, являющихся предметными переменными, свободно входящими в заключение, или предметными константами, входящими в заключение (назовем такие термы *основными* термами секвенции-заключения);

- если в заключении нет основных термов, то терм t является какой угодно предметной переменной, не входящей в заключение.

Для доказательства равнообъемности исчислений $LqSn$ и LqS нам потребуются следующие определение и лемма.

Определение 2.3.5.1. Будем говорить, что дерево (поиска) вывода обладает *свойством уникальности собственных переменных* или является деревом (поиска) вывода с *уникальными собственными переменными*, если для каждого применения плюс-правила его собственная переменная может входить в это дерево только выше заключения данного применения.[6]

Лемма 2.3.5.2. *Если секвенция S выводима в исчислении LqS, то найдется дерево вывода секвенции S в исчислении LqS, обладающее свойством уникальности собственных переменных.*

[5] См. определение 2.3.4.5 на с. 58.

[6] Определенное нами понятие накладывает на вывод меньше ограничений, чем понятие вывода, обладающего свойством чистоты переменных (см. определение последнего понятия, например, в [23], с. 406), несмотря на то, что мы используем только чистые секвенции.

Доказательство. Пусть T — дерево вывода секвенции S. Используем индукцию по числу применений плюс-правил в T, собственные переменные которых нарушают свойство уникальности собственных переменных.

База индукции. Если применений плюс-правил, собственные переменные которых нарушают требуемое свойство, в T нет, то T и является деревом вывода секвенции S, обладающим свойством уникальности собственных переменных.

Индукционный переход. Выберем в T такое применение APR плюс-правила, собственная переменная y которого нарушает требуемое свойство, причем выше этого применения в T нет ни одного применения плюс-правила, собственная переменная которого нарушает требуемое свойство.

Выберем любую новую предметную переменную w, не входящую в T. Заменим все вхождения переменной y на w в посылке применения APR и всюду выше этой посылки. В силу того, что w — новая переменная, получившееся в результате этой замены дерево, очевидно, является деревом вывода секвенции S, причем в нем меньше применений плюс-правил, собственные переменные которых нарушают требуемое свойство, чем в T. Применяя индукционное предположение, получаем дерево вывода секвенции S, обладающее свойством уникальности собственных переменных. □

Теорема 2.3.5.3 (Равнообъемность исчислений $LqSn$ и LqS)**.** *Пусть S — секвенция исчисления LqS. Тогда S выводима в $LqSn$, если и только если S выводима в LqS.*

Доказательство. Необходимость очевидна. Докажем достаточность.

Пусть T — дерево вывода секвенции S в исчислении LqS. Покажем, что T может быть преобразовано в дерево вывода секвенции S в исчислении $LqSn$. В силу леммы 2.3.5.2 будем считать, что T обладает свойством уникальности собственных переменных.

Воспользуемся индукцией по числу тех применений минус-правил в дереве вывода T, которые нарушают ограничение (4) на кванторные правила.

База индукции. Если применений минус-правил, которые нарушают

ограничение (4), нет, то T является также и деревом вывода секвенции S в исчислении $LqSn$.

Индукционный переход. Выберем такое применение AMR минус-правила, которое нарушает ограничение (4), и выше которого в дереве T нет применений минус-правил, нарушающих ограничение (4). Пусть это ограничение нарушается подстановкой терма t в посылку выбранного применения правила. Если в заключении выбранного применения правила нет основных термов, то пусть t' — предметная переменная, не входящая в T, иначе пусть t' — один из основных термов этого заключения. Заменим в посылке выбранного применения минус-правила и в дереве T выше этой посылки все вхождения t на t'. Обозначим полученное с помощью описанной замены дерево посредством T'.

Заметим, что t' — новая переменная, не входящая в T, переменная, входящая свободно в заключение применения AMR минус-правила в T, или предметная константа. Так как мы имеем дело только с чистыми секвенциями, то описанная замена t на t' является свободной подстановкой в посылку применения AMR минус-правила в T и результат этой подстановки является чистой секвенцией. Секвенции, расположенные выше этой посылки, содержат только те связанные переменные, которые содержит эта посылка (это свойство, очевидно, обеспечивают правила вывода[7]), следовательно, утверждение предыдущего предложения верно не только для этой посылки, но и для всех секвенций, расположенных выше нее.

Легко видеть, что при замене t на t' применения правил вывода остались таковыми, кроме разве лишь применений плюс-правил вывода, расположенных выше выбранного ранее применения AMR минус-правила. При применении каждого из плюс-правил требуется, чтобы собственная переменная не входила свободно в заключение. Поэтому достаточно показать, что t' не совпадает ни с какой собственной переменной применений плюс-правил, расположенных выше применения AMR минус-правила.

Отметим, что t', если является предметной переменной, то является

[7]См. также замечание 2.2.1.16 о свойстве подформульности исчисления на с. 45.

либо (a) новой переменной, не входящей в T, либо (b) переменной, входящей свободно в заключение применения AMR минус-правила в T. В случае (a) t' не совпадает ни с какой собственной переменной применений плюс-правил в T и, значит, в T'. В случае (b) в силу того, что T обладает свойством уникальности собственных переменных, t' не может совпадать ни с какой собственной переменной применений плюс-правил, расположенных выше применения AMR минус-правила в T и, значит, в T'.

Корнем дерева T' является та же секвенция S, поскольку заключение применения AMR минус-правила в T такое же, как и в T' (это заключение может совпадать с корнем или находиться выше него).

Теперь для установления того, что T' является деревом вывода секвенции S, достаточно показать, что при замене t на t' аксиомы, находящиеся выше применения AMR минус-правила в T, остаются аксиомами. Пусть S_{leaf} — такая аксиома дерева вывода T, а S'_{leaf} — секвенция дерева T', полученная из S_{leaf} с помощью указанной замены.

Ясно, что терм t' может входить в S_{leaf}, и замена t на t' может лишь увеличить число вхождений t' в S'_{leaf} по сравнению с S_{leaf}, причем в S'_{leaf} нет вхождений t: все они заменены на t'. Поэтому эта замена может только привести к отождествлению некоторых рациональнозначных переменных в системе линейных неравенств, соответствующей секвенции S_{leaf} (см. описание системы неравенств в разделе 2.2.2), т.е. система неравенств, соответствующая секвенции S'_{leaf}, получается из системы неравенств, соответствующей секвенции S_{leaf}, путем отождествления нескольких рациональнозначных переменных.

Соответствующая секвенции S_{leaf} система неравенств несовместна, поскольку S_{leaf} — аксиома. Отождествление рациональнозначных переменных r_1 и r_2 в системе неравенств эквивалентно добавлению двух неравенств $r_1 \leq r_2$ и $r_1 \geq r_2$ в эту систему; следовательно, несовместная система остается несовместной после отождествления некоторых рациональнозначных переменных. Значит, S'_{leaf} является аксиомой. Таким образом, каждый лист дерева T' является аксиомой.

В итоге получено T' — дерево вывода секвенции S, в котором число тех применений минус-правил, которые нарушают ограничение (4) на кванторные правила, меньше, чем в дереве T. Применяя к T' индукционное предположение, получаем дерево вывода секвенции S, в котором нет применений минус-правил, нарушающих ограничение (4), т.е. получаем дерево вывода секвенции S в исчислении $LqSn$.

□

Замечание 2.3.5.4. Рассуждения, подобные приведенным в предыдущей теореме 2.3.5.3, могут использованы для обоснования равнообъемности безантецедентного секвенциального исчисления LS (для классической двузначной логики, см. раздел 2.3.3) и его минус-нормального варианта. Другим способом установления равнообъемности этих исчислений является использование теорем 2.3.3.2 и 2.3.5.3.

Замечание 2.3.5.5. Нетрудно проследить, что все доказанные в этой главе утверждения, касающиеся исчисления LqS, остаются верными и для минус-нормального исчисления $LqSn$.

2.4. Подлогика двучленных нечетких неравенств

Опишем подлогику $Lq2$ логики Lq. В формулах этой подлогики использование логической связки $\prec$ ограничено. Для краткости упомянем только отличия между этими логиками.

Определим язык логики $Lq2$.

Формулой без неравенств назовем любую атомарную формулу, а также формулы $(A\&B)$, $(A \vee B)$, $q \cdot A$, $\forall x A$, $\exists x A$, где A, B — формулы без неравенств, q — рациональное число, x — предметная переменная.

Нечеткое неравенство имеет вид $(A \prec B)$, где A, B — формулы без неравенств.

Формулой логики $Lq2$ является формула без неравенств, нечеткое неравенство, а также формулы $(A\&B)$, $(A \vee B)$, $q \cdot A$, $\forall x A$, $\exists x A$, где A, B — формулы без неравенств, q — рациональное число, x — предметная переменная.

Секвенциальное исчисление $Lq2S$ (соответственно, $Lq2Sn$) для логики $Lq2$ формулируется аналогично исчислению LqS (соответственно, $LqSn$). Чтобы получить исчисление $Lq2S$ (соответственно, $Lq2Sn$) из исчисления LqS (соответственно, $LqSn$), исключим все правила вывода, в которых записи которых фигурирует $\prec_-$, и заменим F_1 and F_2 в записи каждого из оставшихся правил на пустое слово.

Отметим, что из-за ограничения на использование $\prec$ в формулах логики $Lq2$ в исчислении $Lq2S$ КНЦ (см. определение 2.2.2.1 на с. 46) может иметь только один из следующих видов: P, $q \cdot P$ или $(A \prec B)$, где P — атомарная формула, $q\cdot$ — модератор, A, B — формулы вида P или $q \cdot P$ (т.е. в КЦН может быть не более одного вхождения связки $\prec$). Поэтому распознавание аксиомы исчисления $Lq2S$ сводится (см. раздел 2.2.2) к проверке несовместности системы строгих и нестрогих линейных неравенств, каждое из которых имеет не более двух членов. Для решения таких задач существует сильно полиномиальный алгоритм (см. раздел 1.1.4, а также главу 5), тогда как для распознавания аксиом исчисления LqS сильно полиномиальный алгоритм не известен в настоящее время (см. конец раздела 2.2.2). Таким образом, исчисление $Lq2S$ вычислительно эффективнее исчисления LqS.

С другой стороны, логика $Lq2$ может оказаться достаточно выразительной в некоторых случаях, особенно при первоначальном моделировании областей нечетких знаний, когда по сути требуется выразить частичный порядок на множестве некоторых нечеткий утверждений (фактов) в соответствии с оценками истинности этих утверждений. Этим и объясняется выделение подлогики $Lq2$.

Глава 3.

Алгоритм поиска вывода

В этой главе приводится алгоритм поиска вывода[1] *в секвенциальном исчислении* LqS *для расширения* Lq *бесконечнозначной предикатной логики Лукасевича. Доказаны свойства предложенного алгоритма, отражающие различные аспекты его корректности.*

3.1. Идея алгоритма и основные определения

Задачу поиска вывода секвенции в исчислении LqS можно заменить на ту же задачу, но для исчисления $LqSn$ в силу равнообъемности этих исчислений (см. теорему 2.3.5.3). Для того, чтобы попытаться найти вывод секвенции в исчислении $LqSn$, будем строить дерево поиска вывода этой секвенции снизу вверх (см. обсуждение такого способа в начале раздела 2.3.5). Несмотря на то, что мы устранили бесконечный перебор термов для подстановки при контрприменениях минус-правил (рассматривая исчисление $LqSn$ вместо исчисления LqS), задача удачного подбора термов,

[1] В данной работе выражение «алгоритм поиска вывода» понимается буквально — как алгоритм, который может применяться для поиска вывода объектов в рассматриваемом исчислении. Употребление этого выражения не означает, что упоминаемый алгоритм обладает какими-либо дополнительными свойствами. Свойства такого алгоритма будут оговариваться отдельно. (Ср., например, [32], где выражение «алгоритм поиска вывода» подразумевает некоторые свойства алгоритма, и [1], где это выражение используется буквально, без наложения дополнительных требований на упоминаемый алгоритм.)

хотя и упростилась, но все равно остается.

Одним из прямолинейных подходов мог бы быть следующий. В момент контрприменения минус-правила вместо текущего дерева поиска вывода появляется столько деревьев вывода, сколько термов можно подставить вместо переменной, связанной квантором, который вводит это правило.

Мы воспользуемся более экономным подходом, называемом *методом метапеременных* (см., например, [34]). Пополним исходное множество предметных переменных бесконечным разрешимым множеством так называемых *метапеременных*; множество метапеременных не должно пересекаться с исходным множеством предметных переменных. Для определенности будем считать, что метапеременные являются словами, начинающимися со знака '?', за которым следует какое угодно слово из букв английского алфавита и десятичных цифр. Метапеременные удобно считать особым видом предметных переменных, но метапеременные могут использоваться только в ходе поиска вывода и не могут встречаться в секвенции, вывод которой ищется.

Вместо того, чтобы при контрприменении минус-правила подставлять конкретный терм, отложим его выбор, подставив вместо него метапеременную, отличную от каждой из встречающихся в текущем дереве метапеременных. Тем самым вместо дерева поиска вывода строится так называемая *заготовка вывода*, которая отличается от дерева поиска вывода только тем, что в нее могут входить метапеременные. Позднее в ходе поиска вывода, когда, скорее всего, будет больше шансов для удачного выбора термов (если удачный выбор вообще возможен), попытаемся подобрать термы-значения для метапеременных (иначе говоря, конкретизировать метапеременные) так, чтобы превратить заготовку вывода в дерево вывода.

Вспомним, что при поиске вывода достаточно считать, что подставляемый при контрприменении минус-правила терм может принадлежать только конечному множеству термов, задаваемому ограничением (4) на минус-правила (см. это ограничение на с. 64). Поэтому с каждой появляющейся в ходе поиска вывода метапеременной будем ассоциировать конечное мно-

жество термов, содержащее все термы, только которые и достаточно подставлять вместо этой метапеременной в соответствии с ограничением (4) на минус-правила. Тогда для того, чтобы конкретизировать метапеременные с целью превращения заготовки вывода в дерево вывода, можно осуществить (конечный) перебор термов из упомянутых множеств, ассоциированных с каждой метапеременной заготовки вывода, и выбрать те значения метапеременных, при которых заготовка превращается в дерево вывода (точнее, в дерево вывода исходной секвенции в исчислении LqS). Будем называть такой процесс подбора термов-значений для метапеременных с целью превращения заготовки вывода в дерево вывода *унификацией*.

Определение 3.1.0.6. Пусть $Skeleton$ — заготовка вывода, D — множество всех метапеременных, входящих в $Skeleton$, θ — отображение из множества D в множество термов, причем выполняются два условия:

(1) для каждой метапеременной $mv \in D$, указано непустое конечное множество термов $SubstSet(mv)$ (называемое *подстановочным множеством* метапеременной mv), и

(2) для любой метапеременной $mv \in D$ выполняется $\theta(mv) \in SubstSet(mv)$.

Тогда назовем отображение θ *унификатором* заготовки вывода $Skeleton$, если при замене в $Skeleton$ каждого вхождения метапеременной $mv \in D$ на терм $\theta(mv)$ заготовка $Skeleton$ превращается в дерево вывода в исчислении LqS. Если в заготовке вывода нет метапеременных, и эта заготовка является деревом вывода, то будем говорить, что для нее существует пустой унификатор.

Замечание 3.1.0.7. В дальнейшем под заготовкой вывода будем понимать заготовку, для каждой метапеременной mv которой указано подстановочное множество $SubstSet(mv)$ этой метапеременной (см. условие (1) в предыдущем определении).

Определение 3.1.0.8. Заготовку вывода, для которой существует унификатор (в том числе и пустой), назовем *унифицируемой*. В противном случае назовем ее *неунифицируемой*.

Определение 3.1.0.9. Заготовку вывода назовем *существенно неунифицируемой*, если некоторая листовая секвенция этой заготовки совпадает со своей базовой подсеквенцией и не является аксиомой.

Лемма 3.1.0.10. *Пусть S — секвенция заготовки вывода $Skeleton$. Если S совпадает со своей базовой подсеквенцией, то в S не входит ни одна метапеременная.*

Доказательство. Предположим, что в S входит некоторая метапеременная. Рассмотрим ветвь $Branch$ дерева $Skeleton$, ведущую из корня в S. На этой ветви обязательно встретится секвенция, которая является посылкой контрприменения минус-правила (иначе в S, очевидно, не входила бы никакая метапеременная). Выберем на ветви $Branch$ самую близкую к S секвенцию S_e, являющуюся посылкой контрприменения минус-правила. В секвенцию S_e входит квантор, поскольку главная формула минус-правила является членом посылки. В силу выбора секвенции S_e на ветви $Branch$ от S_e до S могут производиться контрприменения только пропозициональных правил вывода. Все такие контрприменения сохраняют упомянутый квантор в каждой своей посылке (в этом можно убедиться, просмотрев все пропозициональные правила вывода). Таким образом, в S входит квантор, и потому S не совпадает со своей базовой подсеквенцией. □

Предложение 3.1.0.11. *Если заготовка существенно неунифицируема, то она неунифицируема.*

Доказательство. Пусть S — листовая секвенция этой заготовки, совпадающая со своей базовой подсеквенцией и не являющаяся аксиомой. Такая секвенция найдется, поскольку заготовка существенно неунифицируема. По лемме 3.1.0.10 в секвенцию S не входит ни одна метапеременная.

Следовательно, ни при каком выборе значений для метапеременных эта заготовка не может являться деревом вывода. □

Замечание 3.1.0.12. Мы условились считать метапеременные особым видом предметных переменных. Это позволяет нам продолжать использовать такие термины, как «секвенция», даже если в нее входят метапеременные.

Однако, употребляя в дальнейшем термин «(предметная) переменная», мы понимаем под ним предметную переменную из множества предметных переменных, не пополненного множеством метапеременных.

3.2. Описание шагов алгоритма

3.2.1. Алгоритм $IsAxiom$

Определим вспомогательный алгоритм $IsAxiom$, которому на вход подается секвенция.

(1) Строится базовая подсеквенция входной секвенции.

(2) Если базовая подсеквенция оказалась пуста, то выдается ответ[2] «не аксиома».

(3) Если базовая подсеквенция не пуста, то выполняются следующие действия.

(3.1) Строится система линейных неравенств, соответствующая этой базовой подсеквенции (построение описано в разделе 2.2.2), и эта система подается на вход алгоритма $IsInconsistent$ проверки несовместности систем линейных неравенств (такой алгоритм указан также в разделе 2.2.2).

[2]Здесь и далее считаем, что выдача алгоритмом ответа означает также завершение работы этого алгоритма.

(3.2) Алгоритм $IsAxiom$ выдает ответ «аксиома», если алгоритм $IsInconsistent$ выдал ответ «несовместна», и «не аксиома», если алгоритм $IsInconsistent$ выдал ответ «совместна».

3.2.2. Алгоритм $Unify$

Определим вспомогательный алгоритм $Unify$, которому на вход подается заготовка вывода.[3]

(1) Если метапеременных в этой заготовке нет, то выполняются следующие действия.

(1.1) Для каждой листовой секвенции заготовки вызывается алгоритм $IsAxiom$.

(1.1.1) Если некоторая листовая секвенция S совпадает со своей базовой подсеквенцией, и алгоритм $IsAxiom$ для входа S выдал ответ «не аксиома», то сразу же выдается ответ «существенно неунифицируема».

(1.2) Если алгоритм $IsAxiom$ выдал ответ «аксиома» для каждой листовой секвенции, то выдается ответ «унифицируема», иначе выдается ответ «неунифицируема».

(2) Если в этой заготовке есть хотя бы одна метапеременная, то выполняются следующие действия.

(2.1) Последовательно перебираются всевозможные упорядоченные n-ки значений метапеременных (где n — число различных метапеременных в заготовке), причем значения каждой метапеременной берутся из конечного подстановочного множества этой метапеременной.

[3]Заготовка вывода является деревом, узлы которого — секвенции. То, как конкретно представлено это дерево, не важно на данном уровне абстракции. Считаем, что используется одно из представлений дерева.

(2.1.1) При получении очередной упорядоченной n-ки значений они подставляются вместо метапеременных в заготовку вывода.

(2.1.2) Для каждой листовой секвенции заготовки вызывается алгоритм $IsAxiom$.

(2.1.2.1) Если некоторая листовая секвенция S совпадает со своей базовой подсеквенцией, и алгоритм $IsAxiom$ для входа S выдал ответ «не аксиома», то сразу же выдается ответ «существенно неунифицируема».

(2.1.3) Если при очередном выборе упорядоченной n-ки значений n-ки значений алгоритм $IsAxiom$ выдал ответ «аксиома» для каждой листовой секвенции, то выдается ответ «унифицируема», и заодно выдается (как составная часть ответа) конечный список пар (метапеременная, терм-значение).

(2.2) Если же перебор всевозможных упорядоченных n-ок значений метапеременных завершен (следовательно, ранее алгоритм $Unify$ не выдал ответ), то выдается ответ «неунифицируема».

3.2.3. Требования к алгоритму $Tactics$

Считаем, что имеется вспомогательный алгоритм $Tactics$, который по любой поданной на вход заготовке вывода $Skeleton$ выдает ответ одного из трех типов:

I. выдает индикатор того, что следует осуществить контрприменение правила, и вместе с этим индикатором выдает секвенцию-лист S, правило вывода (R), контрприменение которого следует осуществить, и вхождение логического символа в S, которое может быть введено в S применением правила (R), если это действительно можно сделать;

II. выдает индикатор того, что ни к одному листу заготовки $Skeleton$ невозможно осуществить контрприменение никакого правила выво-

да,[4] если и только если это действительно так;

III. выдает индикатор того, что следует попытаться превратить заготовку *Skeleton* в дерево вывода,[5] и вместе с этим индикатором выдает секвенцию-лист S, правило вывода (R) и вхождение логического символа в S, которое может быть введено в S применением правила (R), если это действительно можно сделать.

Любой алгоритм, удовлетворяющий перечисленным требованиям, будем называть *тактикой поиска вывода*.

3.2.4. Главный алгоритм *Prove*

Пусть имеется заготовка вывода *Skeleton*. Будем говорить, что метапеременная *новая* для этой заготовки, если эта метапеременная отлична от каждой из встречающихся в этой заготовке *Skeleton* метапеременных. Будем говорить, что предметная переменная *новая* для этой заготовки, если эта предметная переменная отлична от каждой из встречающихся в заготовке *Skeleton* предметных переменных, в том числе тех предметных переменных, которые входят в подстановочные множества метапеременных заготовки *Skeleton*.

Теперь опишем алгоритм *Prove* поиска вывода секвенции S_0 в исчислении $LqSn$ в соответствии с подходом, намеченным выше.

(1) Создается заготовка вывода — дерево с одним узлом, представляющим секвенцию S_0.

(2) Повторяются следующие действия.

(2.1) Вызывается вспомогательный алгоритм *Tactics* и ему на вход подается текущая заготовка вывода.

[4] В этом случае будем также говорить, что эту заготовку невозможно нарастить.

[5] Иначе говоря, провести унификацию или попытаться закрыть заготовку вывода.

(2.2) Если алгоритм $Tactics$ выдал ответ типа I и выбрал секвенцию-лист S, правило вывода (R) и вхождение логического символа, то строятся секвенции-посылки для контрприменения правила (R) к секвенции S в соответствии с выбранным вхождением логического символа, при этом неоднозначность при построении посылок кванторных правил разрешается следующим образом.

(−) Если правило (R) — минус-правило, то находится новая для текущей заготовки метапеременная mv и ее подстановочное множество:

$$SubstSet(mv) = \Big(\bigcup_{v \text{ — метапеременная, входящая в } S} SubstSet(v) \Big) \bigcup \Big(\bigcup_{t \text{ — основной терм, входящий в } S} \{t\} \Big).$$

Если полученное множество $SubstSet(mv)$ оказалось пустым, то находится новая для текущей заготовки предметная переменная a, и полагается $SubstSet(mv) = \{a\}$. Метапеременная mv используется в посылке правила (R) вместо подбираемого терма.

(+) Если правило (R) — плюс-правило, то находится новая для текущей заготовки предметная переменная, которая используется в качестве собственной переменной в посылке правила (R).

Заготовка вывода достраивается так, что непосредственными потомками узла S становятся узлы, представляющие секвенции-посылки правила (R), и происходит переход к шагу (2.1).

(2.3) Если алгоритм $Tactics$ выдал ответ типа II, то вызывается алгоритм $Unify$, на вход которому подается текущее дерево вывода. Если алгоритм $Unify$ выдал ответ «унифицируема», то алгоритм $Prove$ выдает ответ «выводима» и заодно текущую заготовку вывода, иначе алгоритм $Prove$ выдает ответ «невыводима».

(2.4) Если алгоритм $Tactics$ выдал ответ типа III, то вызывается алгоритм $Unify$, на вход которому подается текущее дерево вывода. Если алгоритм $Unify$ выдал ответ «унифицируема» и заодно выдал конечный список θ пар (метапеременная, терм-значение), то алгоритм $Prove$ выдает ответ «выводима» и заодно выдает текущую заготовку вывода и список θ. Если алгоритм $Unify$ выдал ответ «существенно неунифицируема», то алгоритм $Prove$ выдает ответ «невыводима». Если алгоритм $Unify$ выдал ответ «неунифицируема», то производятся те же действия, что и на шаге (2.2), после чего происходит переход к шагу (2.1).

Замечание 3.2.4.1. Для того, чтобы находить новую для текущей заготовки метапеременную, не просматривая все метапеременные этой заготовки, удобно представлять метапеременные с помощью целых положительных чисел (например, '?1', '?2', '?3', ...) и, когда требуется новая метапеременная, брать метапеременную, соответствующую очередному числу.

Замечание 3.2.4.2. Аналогично тому, как множество предметных переменных было пополнено множеством метапеременных, пополним еще раз множество предметных переменных множеством слов, начинающихся со знака '!', за которым следует любое слово из букв английского алфавита и десятичных цифр. Эти слова не могут встречаться в секвенции, вывод которой ищется, но они могут использоваться в качестве предметных переменных, когда требуется найти новую предметную переменную, так же, как и в предыдущем замечании.

3.2.5. Пример поиска вывода

В этом разделе вместо одноместного предикатного символа $P[0,1]$ будем использовать сокращение P. Найдем вывод формулы $(\forall x P(x) \prec (\forall x P(x)\ \&\ 3 \cdot P(z)))$.

Сначала заготовка вывода состоит только из секвенции (1).

(1) $(\underline{\forall} x P(x) \prec (\forall x P(x)\ \&\ 3 \cdot P(z)))$

Произведем контрприменение минус-правила ($q_p\forall \prec_+$) к подчеркнутому вхождению логического символа в (1). Получим секвенцию (2), содержащую метапеременную $?x1$. Для этой метапеременной сформируем подстановочное множество $SubstSet(?x1) = \{z\}$.

(2) $(\forall x P(x) \prec (\forall x P(x)\ \&\ 3 \cdot P(z))),\ (P(?x1) \prec (\forall x P(x)\ \underline{\&}\ 3 \cdot P(z)))$

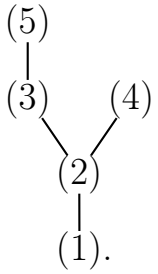
Произведем контрприменение правила ($\prec_+ q_p\&$) к подчеркнутому вхождению логического символа в (2). Получим секвенции (3), (4).

(3) $(\forall x P(x) \prec (\forall x P(x)\ \&\ 3 \cdot P(z))),\ (P(?x1) \prec \underline{\forall} x P(x))$
(4) $(\forall x P(x) \prec (\forall x P(x)\ \&\ 3 \cdot P(z))),\ (P(?x1) \prec 3 \cdot P(z))$

Произведем контрприменение плюс-правила ($\prec_+ q_p\forall$) к подчеркнутому вхождению логического символа в (3). Получим секвенцию (5), содержащую новую собственную переменную $!1$.

(5) $(\forall x P(x) \prec (\forall x P(x)\ \underline{\&}\ 3 \cdot P(z))),\ (P(?x1) \prec P(!1))$

Текущая заготовка вывода имеет вид



Проведем унификацию текущей заготовки вывода с листьями (4) и (5). В текущей заготовке только одна метапеременная $?x1$ с подстановочным множеством $SubstSet(?x1) = \{z\}$. Перебор значений метапеременных сводится к подстановке z вместо $?x1$ в листовые секвенции.

При этой подстановке базовая подсеквенция секвенции (4) имеет вид $(P(z) \prec 3 \cdot P(z))$. Строим систему линейных неравенств, соответствующую этой базовой подсеквенции. Сопоставим рациональнозначную переменную a атомарной формуле $P(z)$. Тогда система такова:

$$3 \cdot a - a < 0,\ 0 \le a,\ a \le 1.$$

Эта система несовместна, поэтому при заданных значениях метапеременных секвенция (4) является аксиомой.

Аналогично действуем с секвенцией (5). Ее базовая подсеквенция при заданной подстановке имеет вид $(P(z) \prec P(!1))$. Рациональнозначные переменные a и b сопоставляем $P(z)$ и $P(!1)$ соответственно. Система неравенств такова:

$$b - a < 0,\ 0 \le a,\ a \le 1,\ 0 \le b,\ b \le 1.$$

Эта система совместна, поэтому при заданных значениях метапеременных секвенция (5) не является аксиомой.

Унификация завершена неудачно, продолжим наращивать заготовку вывода.

Произведем контрприменение правила $(\prec_+ q_p\&)$ к подчеркнутому вхождению логического символа в (5). Получим секвенции (6), (7).

(6) $(\forall x P(x) \prec \underline{\forall} x P(x)),\ (P(?x1) \prec P(!1))$
(7) $(\underline{\forall} x P(x) \prec 3 \cdot P(z)),\ (P(?x1) \prec P(!1))$

Произведем контрприменение правила $(\prec_+ q_p\forall)$ к подчеркнутому вхождению логического символа в (6). Получим секвенцию (8).

(8) $(\underline{\forall} x P(x) \prec P(!2)),\ (P(?x1) \prec P(!1))$

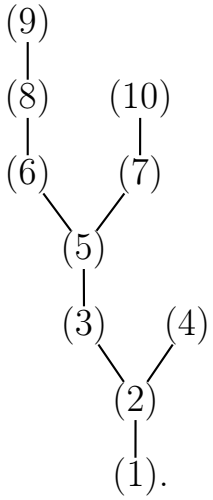
Произведем контрприменение минус-правила $(q_p\forall \prec_+)$ к подчеркнутому вхождению логического символа в (8). Получим секвенцию (9) с новой метапеременной $?x2$, $SubstSet(?x2) = \{!1,\ !2\} \cup SubstSet(?x1) = \{!1,\ !2,\ z\}$.

(9) $(\forall x P(x) \prec P(!2)),\ (P(?x2) \prec P(!2)),\ (P(?x1) \prec P(!1))$

Произведем контрприменение минус-правила $(q_p\forall \prec_+)$ к подчеркнутому вхождению логического символа в (7). Получим секвенцию (10) с новой метапеременной $?x3$, $SubstSet(?x3) = \{z,\ !1\} \cup SubstSet(?x1) = \{z,\ !1\}$.

(10) $(\forall x P(x) \prec 3 \cdot P(z)),\ (P(?x3) \prec 3 \cdot P(z)),\ (P(?x1) \prec P(!1))$

Текущая заготовка вывода имеет вид



Проведем унификацию текущей заготовки вывода с листьями (9), (10) и (4). В заготовку входят метапеременные $?x1$, $?x2$ и $?x3$; их подстановочные множества таковы: $SubstSet(?x1) = \{z\}$, $SubstSet(?x2) = \{!1,\ !2,\ z\}$, $SubstSet(?x3) = \{z,\ !1\}$. На этот раз, перебирая значения метапеременных, получаем, что все листовые секвенции являются аксиомами при $?x1 = z$, $?x2 = !2$ и $?x3 = z$.

В данном небольшом примере можно проверить непосредственно, что заготовка действительно превращается в дерево вывода (а не только листья заготовки становятся аксиомами) при задании указанных значений метапеременным (в частности, такой вопрос корректности алгоритма исследуются в следующем разделе). Таким образом, получен унификатор заготовки, и вывод исходной формулы найден.

3.3. О корректности алгоритма

Предположим, что

- вспомогательный алгоритм $IsConsistent$, проверяющий несовместность системы линейных неравенств (см. раздел 3.2.1), выдает ответ «несовместна», если входная система несовместна, и выдает ответ «совместна», если входная система совместна;

- используется вспомогательный алгоритм $Tactics$, удовлетворяющий предъявленным к нему требованиям (см. раздел. 3.2.3).

При этом предположении исследуем некоторые свойства остальных вспомогательных алгоритмов и главного алгоритма $Prove$.

3.3.1. Свойства алгоритма $IsAxiom$

Предложение 3.3.1.1. *Алгоритм $IsAxiom$ по заданной секвенции S всегда выдает один из ответов: «аксиома», если секвенция S является аксиомой, или «не аксиома», если секвенция S не является аксиомой.*

Доказательство. Если базовая подсеквенция секвенции S пуста, то по определению аксиомы S не является аксиомой; и алгоритм $IsAxiom$ выдает правильный ответ «не аксиома»(см. раздел 3.2.1).

Иначе алгоритм $IsAxiom$ строит систему линейных неравенств, соответствующую базовой подсеквенции секвенции S, и передает эту систему на вход алгоритма $IsInconsistent$. В этом случае правильность ответа алгоритма $IsAxiom$ следует из теоремы 2.2.2.4 и вышеуказанного предположения о корректности алгоритма $IsConsistent$. □

3.3.2. Свойства алгоритма $Unify$

Определение 3.3.2.1. Пусть $Skeleton$ — заготовка вывода. Будем говорить, что $Skeleton$ обладает *свойством уникальности собственных переменных* или является заготовкой вывода с *уникальными собственными переменными*, если для каждого применения плюс-правила его собственная переменная y

- может входить в дерево, которым является заготовка $Skeleton$, только выше заключения данного применения, и
- для каждой метапеременной mv заготовки $Skeleton$: если y совпадает хотя бы с одним термом из $SubstSet(mv)$, то mv может входить в дерево $Skeleton$ только выше этого заключения.

Лемма 3.3.2.2. *Пусть Skeleton — заготовка вывода с уникальными собственными переменными. Пусть также для каждой метапеременной mv этой заготовки подстановочное множество этой метапеременной формируется следующим образом (S — заключение применения плюс-правила, в посылку которого mv подставлена вместо подбираемого терма):*

$$SubstSet(mv) = \Big(\bigcup_{v \text{ — метапеременная, входящая в } S} SubstSet(v) \Big) \bigcup \Big(\bigcup_{t \text{ — основной терм, входящий в } S} \{t\} \Big),$$

если в S входит хотя бы одна метапеременная или хотя бы один основной терм, иначе

$$SubstSet(mv) = \{a\},$$

где a — предметная переменная, не входящая в S.

Тогда при замене каждой метапеременной заготовки Skeleton на какой угодно терм из подстановочного множества этой метапеременной эта заготовка превращается в дерево поиска вывода в исчислении LqS.

Доказательство. Достаточно показать, что при такой замене все секвенции заготовки останутся чистыми, и все ограничения (1)-(3) на кванторные правила будут выполнены (см. с. 43).

Выполнение ограничения (2) гарантируется уникальностью собственных переменных в заготовке.

Для того, чтобы установить чистоту всех секвенций и выполнение остальных ограничений на кванторные правила, достаточно доказать утверждение, сформулированное в следующем абзаце.

Пусть *Skeleton* — заготовка, в корневую секвенцию S_{root} которой могут входить метапеременные, причем выполняется утверждение $PSeq(S_{root})$: для любой метапеременной mv, если mv входит в S_{root}, то никакая переменная из $SubstSet(mv)$ не входит в S_{root} связанно. Тогда для каждой секвенции S заготовки *Skeleton* выполняется утверждение $PSeq(S)$.

Доказательство утверждения из предыдущего абзаца проведем с помощью индукции по числу секвенций в заготовке.

База индукции. Заготовка содержит единственную секвенцию. Эта секвенция является корнем заготовки, поэтому утверждение верно.

Индукционный переход. Рассмотрим корневую секвенцию S_{root} заготовки $Skeleton$. Покажем, что для каждого непосредственного потомка $S_{premise}$ секвенции S_{root} верно утверждение $PSeq(S_{premise})$.

Если S_{root} имеет в заготовке секвенцию-посылку $S_{premise}$, полученную путем контрприменения плюс-правила, то утверждение $PSeq(S_{premise})$ верно. Действительно, в эту посылку могут входить только те связанные переменные, которые входят в S_{root}. А подстановочное множество появившейся в этой посылке метапеременной, если и содержит переменные, то только те, которые

– входят свободно в S_{root},

– не входят в S_{root} вовсе или

– берутся из подстановочных множеств метапеременных, входящих в S_{root} и потому не нарушающих утверждение $PSeq(S_{root})$.

Если же S_{root} имеет в заготовке секвенции-посылки, полученные путем контрприменения правила, отличного от плюс-правила, то утверждение $PSeq(S_{premise})$ (где $S_{premise}$ — любая из упомянутых посылок) верно, поскольку в каждую из этих посылок могут входить только те связанные переменные, которые входят в S_{root}, и новые метапеременные не появляются в этих посылках.

Применяя индукционное предположение к заготовкам, корнями которых являются непосредственные потомки секвенции S_{root}, завершаем доказательство. □

Теорема 3.3.2.3. *Алгоритм $Unify$ по заданной заготовке вывода $Skeleton$, удовлетворяющей условию леммы 3.3.2.2, всегда выдает один из ответов:*

- *«унифицируема» и заодно выдает унификатор заготовки вывода $Skeleton$, представленный в виде конечного (возможно, пустого)*

списка пар (метапеременная, терм-значение), если и только если эта заготовка унифицируема;

- *«существенно неунифицируема», если и только если эта заготовка существенно неунифицируема;*

- *«неунифицируема», если эта заготовка неунифицируема и не является существенно неунифицируемой.*

Доказательство. Пусть в заготовке $Skeleton$ нет метапеременных.

Тогда эта заготовка унифицируема, если и только если каждый лист заготовки $Skeleton$ является аксиомой. Это и проверяет алгоритм $Unify$ (см. раздел 3.2.2) с помощью алгоритма $IsAxiom$, корректность которого уже установлена. Таким образом, алгоритм $Unify$ выдает ответ «унифицируема» (и пустой унификатор), если и только если эта заготовка унифицируема.

Заготовка $Skeleton$ существенно неунифицируема, если и только если в $Skeleton$ найдется листовая секвенция, совпадающая со своей базовой подсеквенцией не являющаяся аксиомой. Алгоритм $Unify$ выдает ответ «существенно унифицируема» в этом и только в этом случае.

Если заготовка $Skeleton$ неунифицируема, но не является существенно неунифицируемой, то не все листовые секвенции заготовки $Skeleton$ являются аксиомами. В этом случае алгоритм $Unify$ выдает правильный ответ «неунифицируема».

Далее рассматривается случай, когда в заготовке $Skeleton$ есть хотя бы одна метапеременная.

По определению унифицируемой заготовки: заготовка $Skeleton$ унифицируема тогда и только тогда, когда найдется отображение θ из множества D всех метапеременных, входящих в заготовку вывода $Skeleton$, в множество термов, причем для любой метапеременной $mv \in D$ выполняется $\theta(mv) \in SubstSet(mv)$, и при замене каждой метапеременной mv в $Skeleton$ на терм $\theta(mv)$ заготовка $Skeleton$ превращается в дерево вывода.

Лемма 3.3.2.2 гарантирует, что в результате замены каждой метапеременной заготовки $Skeleton$ на какой угодно терм из подстановочного множества этой метапеременной заготовка $Skeleton$ превращается в дерево поиска вывода. Дерево поиска вывода является деревом вывода тогда и только тогда, когда все листья этого дерева являются аксиомами.

Алгоритм $Unify$ проверяет аксиоматичность все листья дерева при всевозможных значениях метапеременных из подстановочных множеств. В том и только в том случае, когда при очередном задании значений метапеременным все листья являются аксиомами, алгоритм $Unify$ выдает ответ «унифицируема» и заодно выдает конечный список пар (метапеременная, терм-значение). Таким образом, алгоритм $Unify$ выдает ответ «унифицируема» и заодно выдает унификатор заготовки $Skeleton$, если и только если эта заготовка унифицируема.

Пусть при переборе значений метапеременных в ходе проверки листьев на аксиоматичность обнаруживается листовая секвенция S, совпадающая со своей базовой подсеквенцией и не являющаяся аксиомой. В силу леммы 3.1.0.10 S не содержит метапеременных, и никакая секвенция, содержащая хотя бы одну метапеременную, не совпадает со своей базовой подсеквенцией. Поэтому факт обнаружения такой секвенции S не зависит от значений метапеременных. Следовательно, заготовка $Skeleton$ существенно неунифицируема тогда и только тогда, когда найдется такая секвенция. В этом и только в этом случае алгоритм $Unify$ выдает ответ «существенно неунифицируема».

Если заготовка $Skeleton$ неунифицируема, но не является существенно неунифицируемой, то перебор всех возможных значений метапеременных должен завершиться, и унификатор не должен быть найден. В этом случае алгоритм $Unify$ выдает правильный ответ «неунифицируема». □

3.3.3. Свойства главного алгоритма *Prove*

Теорема 3.3.3.1. *Пусть алгоритм Prove использует любой вспомогательный алгоритм Tactics, который удовлетворяет требованиям, указанным в разделе 3.2.3; S — секвенция, подаваемая на вход алгоритма Prove. Тогда верны следующие утверждения.*

(1) Если алгоритм Prove выдал ответ «выводима», заготовку вывода и конечный (возможно, пустой) список θ пар (метапеременная, терм-значение), то секвенция S выводима в исчислении LqS, и список θ представляет собой унификатор этой заготовки вывода.

(2) Если алгоритм Prove выдал ответ «невыводима», то секвенция S невыводима в исчислении LqS.

(3) Пусть секвенция S не содержит кванторов. Тогда алгоритм Prove выдает ответ «выводима», если секвенция S выводима в исчислении LqS, и выдает ответ «невыводима», если секвенция S невыводима в исчислении LqS.

Доказательство. При производимых алгоритмом *Prove* контрприменениях правил подстановочные множества метапеременных формируются так, как указано в условии леммы 3.3.2.2, и выбор собственных переменных обеспечивает их уникальность, что позволяет опереться на свойства алгоритма $Unify$ (см. теорему 3.3.2.3).

Алгоритм *Prove* выдает ответ «выводима» (а также текущую заготовку и список θ), только если алгоритм $Unify$ выдал ответ, что текущая заготовка «унифицируема» (а также унификатор θ). Таким образом, по теореме 3.3.2.3 о свойствах алгоритма $Unify$ унификатор θ превращает текущую заготовку в дерево вывода секвенции S, следовательно, утверждение (1) выполняется.

Алгоритм *Prove* выдает ответ «невыводима», только если верно одно из двух:

(a) алгоритм $Unify$ выдал ответ, что текущая заготовка «существенно унифицируема»;

(b) алгоритм $Tactics$ выдал ответ, что текущую заготовку невозможно нарастить, и алгоритм $Unify$ выдал ответ, отличный от «унифицируема».

В случае (a) по теореме 3.3.2.3 текущая заготовка существенно унифицируема, т.е. в этой заготовке нашлась листовая секвенция, совпадающая со своей базовой подсеквенцией и не являющаяся аксиомой. Эта листовая секвенция не содержит метапеременных в силу леммы 3.1.0.10. Зададим некоторые значения метапеременным, чтобы получить дерево поиска вывода секвенции S. Тогда по лемме 2.3.1.5 секвенция S невыводима. Итак, в случае (a) утверждение (2) выполнено.

В случае (b) по теореме 3.3.2.3 и предложению 3.1.0.11 текущая заготовка неунифицируема. Так как текущую заготовку невозможно нарастить, то по лемме 2.3.4.10 каждая ее листовая секвенция совпадает со своей базовой подсеквенцией. Поэтому в силу леммы 3.1.0.10 ни одна листовая секвенция не содержит метапеременных. Следовательно, неунифицируемость этой заготовки означает, что не все листья этой заготовки являются аксиомами. Тогда по лемме 2.3.1.4 секвенция S невыводима. Итак, в случае (b) утверждение (2) выполнено.

Требования к алгоритму $Tactics$ гарантируют, что контрприменения правил к листьям текущей заготовки производятся до тех пор, пока контрприменение какого-либо правила можно осуществить. Поэтому для секвенции без кванторов алгоритм $Prove$ действует аналогично алгоритму, описанному в доказательстве теоремы 2.3.4.12, поэтому такое же доказательство годится для обоснования утверждения (3). □

3.3.4. Выбор алгоритма $Tactics$

Отметим, что в дереве вывода в исчислении LqS, вообще говоря, нельзя менять местами два соседних (контр)применения правил вывода. Это

поясняется следующими примерами.

Пусть дано дерево вывода, и некоторое его поддерево (a) с нижней секвенцией $\Delta, ((A\&B) \prec \forall x C(x))$ имеет вид (C — одноместный предикатный символ; справа от секвенции указано правило, заключением применения которого является эта секвенция):

$$\begin{array}{l} \Delta, (A \prec C(y)), (B \prec C(y)) \\ \Big| \\ \Delta, ((A\&B) \prec C(y)) \quad (q_p\& \prec_+) \\ \Big| \\ \Delta, ((A\&B) \prec \forall x C(x)) \quad (\prec_+ q_p\forall). \end{array}$$

Попытаемся сначала осуществить контрприменение правила $(q_p\& \prec_+)$ к секвенции $\Delta, ((A\&B) \prec \forall x C(x))$, а затем — правила $(\prec_+ q_p\forall)$. Получаем поддерево (b):

$$\begin{array}{l} \Delta, (A \prec C(u)), (B \prec C(v)) \\ \Big| \\ \Delta, (A \prec C(u)), (B \prec \forall x C(x)) \quad (\prec_+ q_p\forall) \\ \Big| \\ \Delta, (A \prec \forall x C(x)), (B \prec \forall x C(x)) \quad (\prec_+ q_p\forall) \\ \Big| \\ \Delta, ((A\&B) \prec \forall x C(x)) \quad (q_p\& \prec_+). \end{array}$$

В силу ограничения на применения плюс-правил собственные переменные u и v каждого из таких применений не должны совпадать, поэтому путем выбора подходящих собственных переменных мы не можем получить ту же верхнюю секвенцию, что и в поддереве (a). Обратно, если дано дерево вывода и его поддерево вида (b) с нижней секвенцией $\Delta, ((A\&B) \prec \forall x C(x))$, то, изменив порядок контрприменений правил $(q_p\& \prec_+)$ и $(\prec_+ q_p\forall)$, не получаем ту верхнюю секвенцию, которая является верхней в поддереве (b).

Алгоритм *Tactics* служит для выбора правил вывода для контрприменений. Из сказанного выше следует, что для практического использования алгоритма *Prove* особенно важна возможность задания различных предпочтительных порядков контрприменений правил.

Опишем один из алгоритмов $Tactics$, удовлетворяющий поставленным в разделе 3.2.3 требованиям. Для этого сначала дополним алгоритм $Prove$.

- Дополним п. (2.2) алгоритма $Prove$: при контрприменении минус-правила вывода (R) дополнительно производится пометка вхождения квантора следующим образом. Пусть $\phi_{conclusion}$ — вхождение главной формулы правила (R) в заключение этого правила (эта формула является как членом заключения, так и членом посылки), $\phi_{premise}$ — вхождение этой же формулы в посылку правила (R), Q — вхождение квантора, вводимое правилом (R), в $\phi_{conclusion}$. Если Q в $\phi_{conclusion}$ помечено целым положительным числом n, то Q в $\phi_{premise}$ помечается числом $n + 1$. Если Q в $\phi_{conclusion}$ не помечено целым положительным числом (а изначально так и будет), то Q в $\phi_{premise}$ помечается числом 1.

- Дополним п. (1) алгоритма $Prove$: с текущей заготовкой вывода ассоциируется число $maxExpansions$, сначала равное 1.

Замечание 3.3.4.1. Про упомянутое выше вхождение квантора, вводимое правилом (R), будем также говорить, что оно раскрывается контрприменением правила (R). Число, которым помечено вхождение квантора, назовем числом раскрытий этого вхождения квантора.

Теперь приведем алгоритм $Tactics$.

Алгоритм $Tactics$ последовательно просматривает все листья текущей заготовки.
1. Если нашлось отличное от минус-правила однопосылочное правило, контрприменение которого можно осуществить, то выдается ответ типа I с этим правилом.
2. Иначе, если нашлось двухпосылочное правило, контрприменение которого можно осуществить, то выдается ответ типа I с этим правилом.

3. Иначе, если нашлось минус-правило вывода, контрприменение которого можно осуществить, и вводимое этим правилом вхождение квантора не помечено или помечено числом, меньшим $maxExpansions$, то выдается ответ типа I с этим правилом.

4. Иначе, если нашлось минус-правило вывода, контрприменение которого можно осуществить (при этом вводимое этим правилом вхождение квантора помечено числом, равным $maxExpansions$), то $maxExpansions$ увеличивается на 1 и выдается ответ типа III с этим правилом.

5. Иначе выдается ответ типа II (т.е. нет ни одного правила, контрприменение которого можно произвести).

Следующие эвристические доводы подтверждают целесообразность такого алгоритма $Tactics$. Во-первых, такой алгоритм не вынуждает слишком часто проводить унификацию (без получения новых членов в каждой листовой секвенции унификация зачастую не приводит к нахождению унификатора). Во-вторых, алгоритм предпочитает контрприменения «простых», детерминированных правил контрприменениям более «сложных», недетерминированных правил. Наконец, в-третьих, алгоритм выбирает минус-правила для контрприменения «равномерно», давая равные возможности вхождениям кванторов быть раскрытыми.

В завершение этого раздела подчеркнем, что все установленные свойства главного и вспомогательных алгоритмов опираются лишь на общие требования к алгоритму $Tactics$, и главный алгоритм для поиска вывода разработан так, что он допускает различные варианты алгоритма $Tactics$.

Глава 4.

Программная реализация алгоритма поиска вывода

В этой главе описана реализация алгоритма поиска вывода, приведенного в предыдущей главе, на языке программирования Java в виде интерфейса прикладного программирования (ИПП), или библиотеки. При использовании этого ИПП можно задать любую тактику поиска вывода. Также можно легко изменить правила вывода. Разработанный ИПП может служить ядром какой-либо дедуктивной системы, основанной на логике Lq; и в этом его преимущество перед программой, в которой не выделен программный интерфейс для доступа к ее возможностям. Кроме того, вместе с реализованным синтаксическим анализатором формул этот ИПП является независимым приложением для поиска вывода формул в исчислении LqS.

Изложенный выше алгоритм поиска вывода является принципиальной схемой реализованного на языке программирования Java программного комплекса, обладающего большей алгоритмической декомпозицией. Этот программный комплекс реализован в виде интерфейса прикладного программирования (ИПП), который предоставляет программный интерфейс для доступа к своим функциональным возможностям.

Предполагается знакомство с объектно-ориентированным программи-

рованием (см., например, [8, 6, 45, 42, 38]); унифицированным языком моделирования (см., например, [47, 9]); языком программирования Java, точнее, Java 2 Platform Standard Edition 5.0 (см. [64] и, например, [2]); основными структурами данных (см., например, [5, 10, 26]); а также с основами синтаксического анализа (см., например, [3]).

4.1. Общая структура

ИПП для поиска вывода состоит из 4 пакетов:

- `prover`,
- `prover.calculus`,
- `prover.calculus.rules`,
- `prover.calculus.syntax`.

Для каждого пакета перечислим основные классы и интерфейсы и укажем их назначение.

4.1.1. Пакет `prover`

Класс `Prover`	Заключает в себе главный алгоритм поиска вывода.
Класс `Unification`	Осуществляет унификацию заготовки вывода.
Класс `InstantiationContext`	Представляет отображение множества метапеременных в множество термов-значений (иначе говоря, конкретизацию метапеременных).
Класс `ProofEvent`	Представляет объект (событие), который содержит информацию о текущем шаге поиска вывода.

Интерфейс `ProofListener`	Интерфейс, задающий получателей (слушателей) событий `ProofEvent`.
Интерфейс `Tactics`	Задает тактику поиска вывода.
Класс `DefaultTactics`	Реализует интерфейс `Tactics`, делая выбор так, как это делает алгоритм *Tactics*, описанный в разделе 3.3.4.
Класс `ProofSearchOptions`	Служит для задания опций поиска вывода.
Класс `Parser`	Производит синтаксический анализ строки, представляющей секвенцию.
Класс `UtilFactory`	Экземпляр-одиночка (см. [42]) этого класса создает списки (экземпляры интерфейса[1] `java.util.List`), множества (экземпляры интерфейса `java.util.Set`), отображения[2] (экземпляры интерфейса `java.util.Map`), которые используются как вспомогательные структуры данных в реализации этого ИПП.

4.1.2. Пакет `prover.calculus`

Ниже перечислены основные классы этого пакета.

`Sequent`	Представляет секвенцию.
`InferenceRule`	Абстрактный класс, служащий для представления правила вывода.

[1] Более точно, экземпляры некоторого класса, который сам (или некоторый его суперкласс) реализует указанный интерфейс. Однако приведенное в тексте выражение общеупотребительно.

[2] Отображения также называют ассоциативными массивами.

`AllInferenceRules`	Содержит список всех правил вывода и реализует соответствие между строковыми обозначениями правил вывода и конкретными экземплярами класса `InferenceRule`.
`ProofSkeleton`	Представляет заготовку вывода.
`AxiomRecognizer`	Служит для проверки того, является ли какая-либо секвенция аксиомой.
`AxiomRecognizer.LinearInequalitySystem`	Представляет систему линейных неравенств, которая строится при проверке секвенции на аксиоматичность.

4.1.3. Пакет `prover.calculus.rules`

Все классы этого пакета являются наследниками класса `InferenceRule` и их объекты представляют те правила вывода, которые указаны ниже.

`Connective1PremiseRule`	Однопосылочное пропозициональное правило вывода.
`ModeratorCompositionRule`	Правило представления модератора в виде произведения двух модераторов.
`ModeratorOutOfInequalityRule`	Правило вынесения модератора из нечеткого неравенства.
`Connective2PremiseRule`	Двухпосылочное пропозициональное правило вывода.
`ExistentialQuantifierRule`	Минус-правило (кванторное правило экзистенциального типа).

`UniversalQuantifierRule`	Плюс-правило (кванторное правило универсального типа).

4.1.4. Пакет `prover.calculus.syntax`

Ниже перечислены все классы этого пакета и их назначение.

`LogicalSymbol`	Абстрактный класс — вершина иерархии классов, представляющих логические символы.
`Quantifier`	Служит для представления кванторов.
`LogicalConnective`	Служит для представления логических связок.
`Moderator`	Служит для представления модераторов.
`FuzzyInequality`	Служит для представления нечеткого неравенства.
`Term`	Абстрактный класс — вершина иерархии классов, представляющих термы.
`IndividualConstant`	Представляет предметную константу.
`IndividualVariable`	Абстрактный класс для представления предметных переменных.
`BoundIndividualVariable`	Представляет связанную предметную переменную.
`FreeIndividualVariable`	Представляет свободную предметную переменную.
`Metavariable`	Представляет метапеременную.

`Formula`	Абстрактный класс — вершина иерархии классов, представляющих формулы.
`LogicalSymbolFormula`	Абстрактный класс для представления формулы, корень дерева которой[3] помечен логическим символом.
`QuantifierFormula`	Представляет формулу, корень дерева которой помечен квантором.
`ConnectiveFormula`	Представляет формулу, корень дерева которой помечен логической связкой.
`ModeratorFormula`	Представляет формулу, корень дерева которой помечен модератором.
`FuzzyInequalityFormula`	Представляет формулу, корень дерева которой помечен нечетким неравенством.
`AtomicFormula`	Абстрактный класс для представления атомарных формул.
`TruthConstant`	Представляет истинностную константу.
`PredicateSymbol`	Представляет предикатный символ вместе со списком аргументов.
`PropositionalVariable`	Представляет пропозициональную переменную.
`TruthValueSegment`	Представляет отрезок истинностных значений.
`RationalNumber`	Представляет рациональное число.

[3]См. определение 2.1.1.7 на с. 32.

`LogicalFactory`	Экземпляр-одиночка этого класса создает экземпляры логических символов, переменных, формул.

Отношение наследования между классами пакета `prover.calculus.syntax` и их внутреннюю структуру наглядно представляют следующие диаграммы классов:

- диаграмма классов, изображающая иерархию логических символов (рис. 4.1 на с. 100);
- диаграмма классов, изображающая иерархию термов (рис. 4.2 на с. 101);
- диаграмма классов, изображающая иерархию формул (рис. 4.3 на с. 102).

Замечание 4.1.4.1. На этих диаграммах показаны не все методы. Наиболее важные методы будут упомянуты ниже при обсуждении декомпозиции алгоритма поиска вывода.

4.2. Представление формул и секвенций

Формула представляется в виде синтаксического дерева (см., например, [3]). Внутренними узлами такого дерева являются экземпляры конкретных потомков класса `LogicalSymbolFormula`, листьями — экземпляры конкретных потомков класса `AtomicFormula`. По сути, синтаксическое дерево формулы (без привязки к программному представлению) является деревом формулы (см. определение 2.1.1.7 на с. 32).

Секвенция (объект класса `Sequent`) представлена списком формул (`java.util.List<Formula>`).

Замечание 4.2.0.2. В стандартном библиотечном пакете `java.util` есть несколько классов, реализующий этот интерфейс, в частности, класс, ос-

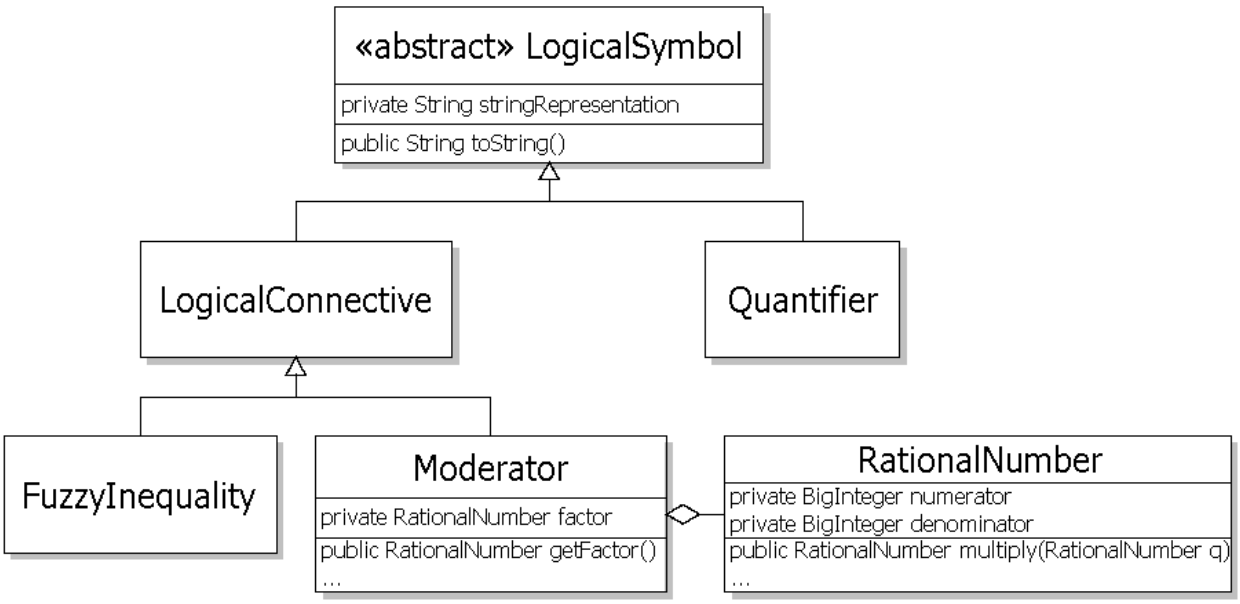


Рис. 4.1. Диаграмма классов, изображающая иерархию логических символов.

нованный на представлении списка с помощью массива, и класс, основанный на ссылочном представлении. Вспомним, класс `UtilFactory` служит для создания объектов, представляющих основные структуры данных. В остальном программном коде используются интерфейсные типы для этих структур данных. Поэтому код легко сконфигурировать, заменив только в классе `UtilFactory` одну реализацию интерфейса на другую.

Требование чистоты любой секвенции (см. определение 2.2.0.5 на с. 37) распространяется и на программное представление секвенции.

В алгоритме поиска вывода фигурирует понятие вхождения логического символа в формулу и секвенцию. Вхождение логического символа в формулу однозначно определяется соответствующим внутренним узлом синтаксического дерева этой формулы. Вхождение логического символа ls в секвенцию однозначно определяется вхождением ls в формулу-член этой секвенции, если эта формула может быть однозначно идентифицирована в секвенции.

В связи с этим на программное представление любой секвенции S наложим требование *корректности разделения*:

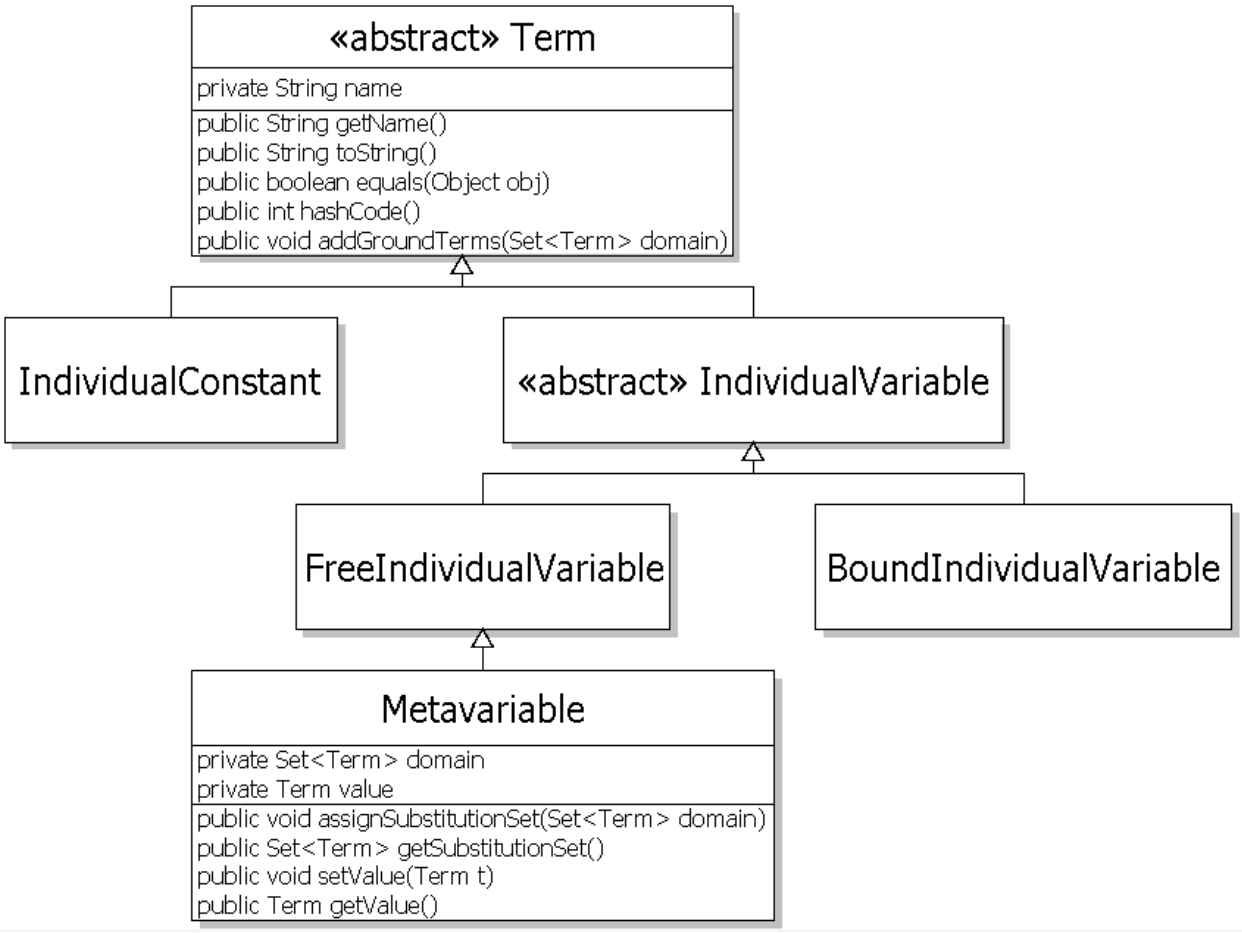


Рис. 4.2. Диаграмма классов, изображающая иерархию термов.

Рис. 4.3. Диаграмма классов, изображающая иерархию формул.

- объектные ссылки любых двух членов секвенции S должны быть различны, и

- для каждой формулы-члена секвенции S в синтаксическом дереве этой формулы нет разделяемых внутренних узлов (иначе говоря, в направленном ациклическом графе, представляющем формулу, только вершины, представляющие атомарные формулы, могут иметь более одной входящей дуги).

Пример 4.2.0.3. Рассмотрим формулу $((P[0,1] \vee 1) \prec (P[0,1] \vee 1))$. На рисунках 4.4 (с. 103) и 4.5 (с. 104) изображены диаграммы объектов, представляющие синтаксические деревья этой формулы. Оба эти представления обладают свойством корректности разделения. Но узел, представляющий неатомарную подформулу $(P[0,1] \vee 1)$, не может быть разделяемым.

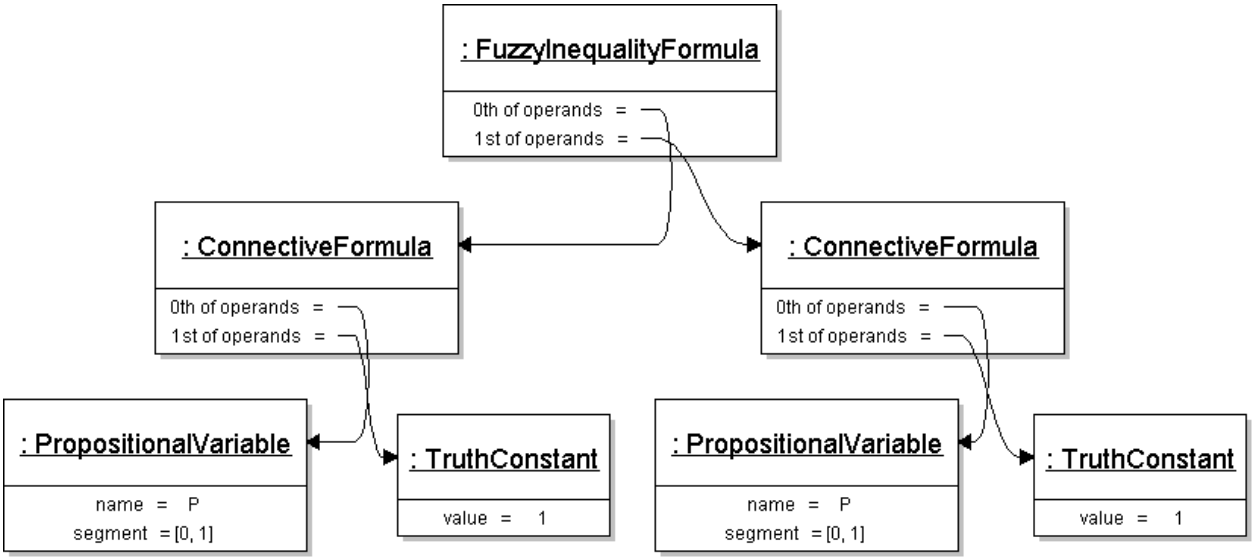


Рис. 4.4. Диаграмма объектов: дерево формулы без разделяемых узлов.

Итак, корректность разделения позволяет однозначно идентифицировать по объектной ссылке

- формулу-член секвенции среди других членов этой секвенции, и

- неатомарную подформулу в формуле-члене секвенции среди других неатомарных подформул этой секвенции.

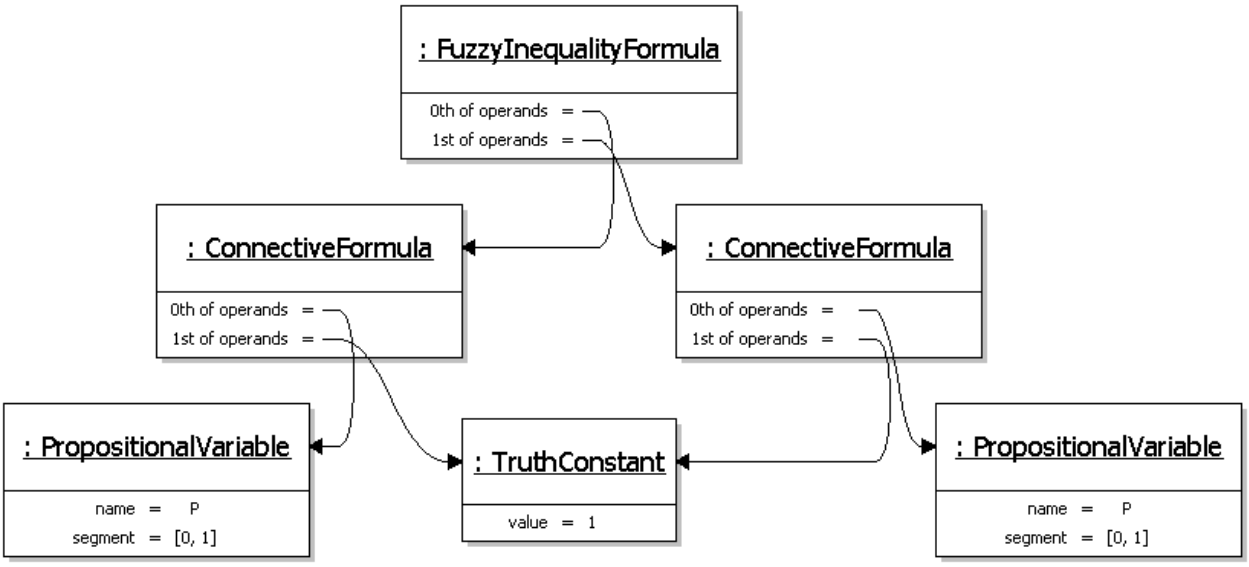


Рис. 4.5. Диаграмма объектов: дерево формулы с (некоторыми) разделяемыми листьями.

Классы `Formula` и `Sequent` имеют методы, проверяющие их чистоту и требование корректности разделения.

Отметим, что различные секвенции могут разделять объекты, представляющие формулы-члены и подформулы этих секвенций. Как мы знаем, при поиске вывода создаются новые секвенции-посылки на основе секвенции-заключения некоторого правила вывода. Посылки имеют много общих формул-членов с заключением, и ссылки на эти формулы могут быть перенесены без копирования из заключения в посылки. Таким образом, в рамках определенной нами политики разделения объектов при поиске вывода происходит значительная экономия памяти.

4.3. Синтаксический анализатор

В ИПП для поиска вывода входит синтаксический анализатор секвенций (класс `Parser`). Этот анализатор по строке (объекту класса `java.lang.String`) строит синтаксическое дерево для каждой формулы-члена секвенции и саму секвенцию, представляемую списком этих деревьев.

Логические символы в строке кодируются следующим образом: `&` (конъюнкция), `V` (дизъюнкция), `<` (нечеткое неравенство), `A` (квантор всеобщности), `E` (квантор существования); знак $\cdot$ в модераторе кодируется как `*`.

Секвенция и формула задаются следующей формой Бэкуса-Наура. (Терминалы записаны в кавычках.)

```
Секвенция = Формула | Формула "," Секвенция

Формула = ИстинностнаяКонстанта | ПредикатныйСимвол |
    ПредикатныйСимвол "(" СписокТермов ")" |
    "(" Формула "&" Формула ")" | "(" Формула "V" Формула ")" |
    "(" Формула "<" Формула ")" | Модератор Формула |
    "A" ПредметнаяПеременная Формула |
    "E" ПредметнаяПеременная Формула
ИстинностнаяКонстанта = Рациональное
Модератор = Рациональное "*"
НеотрицательноеРациональное = НеотрицательноеЦелое |
    НеотрицательноеЦелое "/" ПоложительноеЦелое
Рациональное =  НеотрицательноеРациональное |
    "-" НеотрицательноеРациональное
ПредикатныйСимвол = "P" БуквыИЦифры |
    "P" БуквыИЦифры "[" Рациональное "," Рациональное "]"
СписокТермов = Терм | Терм "," СписокТермов
Терм = ПредметнаяПеременная | ПредметнаяКонстанта
ПредметнаяПеременная = СтрочнаяБуква БуквыИЦифры
ПредметнаяКонстанта = "#" БуквыИЦифры
```

Нетерминалы `НеотрицательноеЦелое`, `ПоложительноеЦелое`, `БуквыИЦифры` и `СтрочнаяБуква` имеют очевидный смысл, если уточнить, что используются десятичные цифры и буквы английского алфавита. Правила для этих нетерминалов не приводятся в целях сокращения записи.

Лексический анализатор реализован как класс `Lexer`, вложенный в класс `Parser`. Синтаксический анализ производится методом рекурсивного спуска (см., например, [3, 48]).

При синтаксическом анализе выполняется также проверка чистоты входной секвенции. Для этого заводятся 3 множества (их элементами являются строки — имена переменных):

- множество `setBV` связанных предметных переменных, встретившихся в просмотренной части секвенции;
- множество `setFV` свободных предметных переменных, встретившихся в просмотренной части секвенции;
- множество `setBVinScope` предметных переменных, которые связаны в текущей области действия вхождений кванторов, встретившихся в просмотренной части секвенции.

Работа с этими множествами (и тем самым проверка чистоты секвенции) производится следующим образом.

(1) Если очередная просканированная лексическим анализатором лексема — квантор, то сканируется следующая лексема. Она должна быть именем `name` предметной переменной (в противном случае об ошибке в записи секвенции сообщается путем бросания исключения).

  (1.1) Если `setFV` или `setBVinScope` содержит `name`, то секвенция не является чистой (об этом также сообщается путем бросания исключения);

  (1.2) иначе `name` добавляется в `setBVinScope`.

    (1.2.1) Производится анализ подформулы, начинающейся сразу за этим вхождением переменной `name`.

  (1.3) `name` удаляется из `setBVinScope`.

(2) Если очередная просканированная лексическим анализатором лексема — имя `name` предметной переменной, и `setBVinScope` не содержит `name`, то

(2.1) если `setBV` содержит `name`, то секвенция не является чистой (об этом сообщается путем бросания исключения);

(2.2) иначе `name` добавляется в множество `setFV`.

Если анализ секвенции завершен, и в его ходе не было сообщено, что эта секвенция не является чистой, то, нетрудно видеть, эта секвенция чиста.

Похожим образом проверяются на чистоту формулы и секвенции, уже представленные в виде объектов (но операции производятся над синтаксическими деревьями, а не над строкой).

Отметим, что вышеприведенная форма Бэкуса-Наура задает также то, как объекты, представляющие формулы и секвенции, преобразуются в строки с помощью метода `toString()`, который переопределен в соответствующих классах.

4.4. Детализация алгоритма поиска вывода

Опишем программную реализацию алгоритма поиска вывода. При этом сосредоточимся на алгоритмической и объектной декомпозиции, поскольку сами шаги алгоритма поиска вывода подробно описаны в разделе 3.2.

4.4.1. Заготовка вывода

Заготовка вывода представляется объектом класса `ProofSkeleton`; узел заготовки — объектом класса `ProofSkeleton.Node`. В заготовке хранится текущий список листьев и входящих в эту заготовку метапеременных.

Узел хранит секвенцию, ссылку на родительский узел и список ссылок на непосредственных потомков, а также информацию о контрприменении правила к этому узлу (такая информация представляется объектом класса `Tactics.RuleChoice`, речь о котором пойдет дальше). В момент созда-

ния узла его потомки неизвестны. Потомки появляются при контрприменении правила вывода к секвенции, находящейся в узле. Поэтому в классе `ProofSkeleton.Node` определен метод

```
public void createChildren(List<Sequent> childSequents,
                           Tactics.RuleChoice ruleChoice)
```

для создания непосредственных потомков узла.

При создании объекта-заготовки вывода можно указать, следует ли экономить память, храня только листья этой заготовки, либо следует хранить все узлы. По умолчанию хранятся только листья заготовки.

4.4.2. Тактики поиска вывода

Вспомним (см. раздел 3.2.3), что для контрприменения правила к заготовке тактика поиска вывода выбирает секвенцию-лист S, правило вывода (R), контрприменение которого следует осуществить, и вхождение логического символа в S, которое может быть введено в S применением правила (R). Учитывая требование корректности разделения для секвенций (см. раздел 4.2), представим этот выбор так:

(a) ссылка объект, представляющий правило вывода (R),

(b) ссылка на лист заготовки вывода с хранящейся в нем секвенцией S,

(c) ссылка на главную формулу F_m правила (R) (F_m — член секвенции S) и

(d) ссылка на подформулу F_s главной формулы F_m, причем корень дерева этой подформулы помечен логическим символом, который правило (R) вводит.

В ИПП для задания тактик поиска вывода служит интерфейс `Tactics`. Только что обсуждавшийся выбор правила для контрприменения представляется объектом класса `Tactics.RuleChoice` со следующими полями (приведенными в порядке (a), (b), (c), (d)):

```
private InferenceRule rule;
private ProofSkeleton.Node node;
private Formula mainFormula;
private Formula subformula
```

(и методами доступа к этим полям, например, `getRule()`).

Тактика также дает указания о том, когда следует попытаться закрыть текущую заготовку, проведя унификацию. Это указание представляется одной из констант перечисления (enum) `Tactics.Direction`. Ниже записаны константы этого перечисления в порядке, соответствующем порядку типов ответов алгоритма *Tactics* (см. раздел 3.2.3):

```
APPLY_RULE,
CLOSE_SKELETON_NO_APPLICABLE_RULE,
CLOSE_SKELETON_APPLICABLE_RULE_EXISTS.
```

Полный ответ тактики представляется объектом класса `Tactics.NextStep` со следующими полями:

```
private Direction direction;
private RuleChoice ruleChoice
```

(и методами доступа к этим полям, например, `getDirection()`). Такой ответ возвращается методом `Tactics.chooseNextStep()`.

Главный алгоритм поиска вывода *Prove*, описанный в разделе 3.2.4, реализован в виде метода `prove()` класса `Prover`. Этот метод в цикле запрашивает указание от тактики и действует в соответствии с этим указанием, осуществляя контрприменение правила или проводя унификацию. Контрприменение правила и унификация заслуживают более подробного рассмотрения в следующих разделах.

Здесь же отметим, что такая архитектура (соответствующая образцу проектирования *стратегия* [42]) с выделением интерфейса для задания тактики позволяет использовать разные тактики, не затрагивая остальные части алгоритма поиска вывода. Наглядно эта архитектура изображена с помощью диаграммы классов на рис. 4.6 (с. 110).

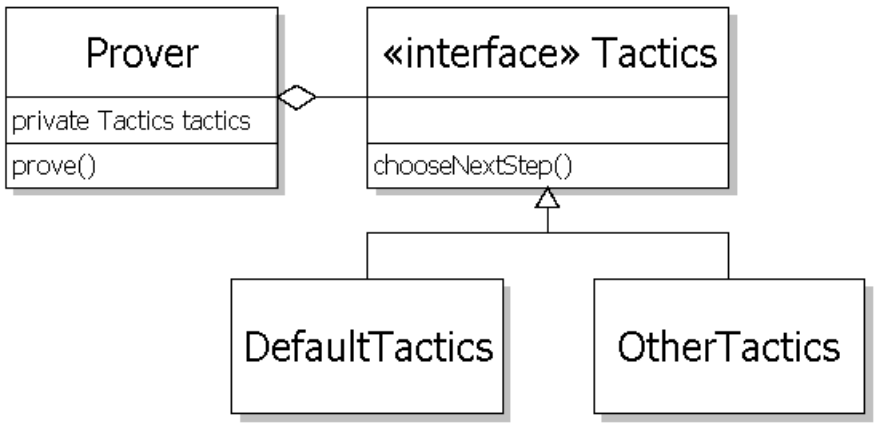


Рис. 4.6. Диаграмма классов, изображающая архитектуру с выделенной тактикой поиска вывода.

Класс `OtherTactics`, изображенный на этой диаграмме, — любая реализация интерфейса `Tactics`, такой класс может быть предоставлен пользователем ИПП для поиска вывода.

Класс `DefaultTactics`, изображенный на этой диаграмме, упоминался в разделе 4.1.1, он реализует тактику поиска вывода так, как это делает алгоритм *Tactics*, описанный в разделе 3.3.4. Вспомним, что `DefaultTactics` отслеживает числа раскрытий вхождений кванторов (см. раздел 3.3.4). Для этого используются методы

```
public Object getLabel();
public void setLabel(Object lbl)
```

класса `Formula`. Первый метод выдает пометку формулы, второй — помечает формулу объектом, указанным в качестве параметра. `DefaultTactics` помечает формулу, если в корне синтаксического дерева этой формулы находится квантор, и этот квантор вводится минус-правилом, выбранным для контрприменения. Эти методы не предназначены только для реализации `DefaultTactics`, а могут применяться в других целях, например, в реализации другой тактики поиска вывода.

4.4.3. Контрприменение правила вывода

Главный алгоритм поиска вывода, которой реализован в виде метода `Prover.prove()`, делегирует контрприменение правила вывода самому выбранному правилу. Таким образом, правила вывода могут быть легко изменены.

Каждое правило вывода представлено объектом одного из 6 классов, перечисленных в разделе 4.1.3. Все эти классы являются наследниками абстрактного класса `InferenceRule`. Основной метод этого класса —

`public abstract void applyBackward(Tactics.RuleChoice ruleChoice).`

Этот метод формирует посылки правила вывода и добавляет к узлу `ruleChoice.getNode()` заготовки эти посылки в качестве непосредственных потомков.

Приведем код метода `applyBackward()` класса `ExistentialQuantifierRule`, представляющего наиболее трудное для контрприменения правило вывода — минус-правило.

```
public void applyBackward(Tactics.RuleChoice ruleChoice) {
    ProofSkeleton.Node node = ruleChoice.getNode();
    Formula mainFormula = ruleChoice.getMainFormula();
    Formula subformula = ruleChoice.getSubformula();
    Sequent sequent = node.getSequent();

    List<Formula> premiseSuccedent =
            UtilFactory.getInstance().createList();

    for (Formula f : sequent.getSuccedent()) {
        if (f == mainFormula) {
            QuantifierFormula qf = (QuantifierFormula)subformula;
            Formula qsf = qf.getQuantifiedSubformula();

            Set<Term> domain = sequent.getGroundTerms();
            if (domain.isEmpty()) {
                FreeIndividualVariable fiv =
                        LogicalFactory.getInstance().
                        generateNewFreeIndividualVariable();
```

```
                domain.add(fiv);
            }
            Metavariable mv =
                    LogicalFactory.getInstance().
                    generateNewMetavariable(qf.getQuantifiedVariable());
            mv.assignSubstitutionSet(domain);
            node.addMetavariable(mv);

            Formula qsfWithSubstitution =
                    qsf.replaceVariable(qf.getQuantifiedVariable(), mv);

            Formula sideFormula = mainFormula.replaceSubformula(qf,
                    qsfWithSubstitution);
            premiseSuccedent.add(sideFormula);
        } // end of: if (f == mainFormula)

        premiseSuccedent.add(f);
    } // end of: for (Formula f : sequent.getSuccedent())

    List<Sequent> premises = UtilFactory.getInstance().createList();
    Sequent premise = new Sequent(premiseSuccedent);
    premises.add(premise);

    node.createChildren(premises, ruleChoice);
}
```

Замечание 4.4.3.1. В программном коде определения методов снабжены документирующими комментариями. Поэтому при работе с программным кодом не возникает проблем понимания того, что выполняет тот или иной метод.

Поясним здесь наиболее существенные детали метода, приведенного выше. Посылка минус-правила формируется из всех формул заключения и боковой формулы (`sideFormula` в коде), содержащей вводимую данным контрприменением метапеременную. Для построения этой боковой формулы сначала находится подстановочное множество для создаваемой метапеременной с помощью метода класса `Sequent`

`public Set<Term> getGroundTerms()`.

Замечание 4.4.3.2. Метод `Sequent.getGroundTerms()` реализован так, что он по очереди вызывает метод класса `Formula`

`public void addGroundTerms(Set<Term> domain)`

для каждой формулы-члена секвенции. Последний метод переопределен в подклассах класса `Formula`, в том числе в `PredicateSymbol`. Объекты класса `PredicateSymbol` служат источниками термов-элементов подстановочного множества. В момент сбора таких термов, входящих в объект класса `PredicateSymbol`, нужна информация о том, является ли каждая встречающаяся переменная связанной. Но поскольку используются отдельные классы для связанных и свободных переменных (соответственно, классы `BoundIndividualVariable` и `FreeIndividualVariable`), то переменная является связанной, если и только если она представлена объектом класса `BoundIndividualVariable`. Таким образом, выбранное представление свободных и связанных переменных отдельными классами обеспечивает эффективность и простоту реализации нахождения подстановочного множества.

Продолжим обсуждение вышеприведенного метода `applyBackward()`. После нахождения подстановочного множества создается новая метапеременная mv (`mv` в коде), и устанавливается ее подстановочное множество.

Замечание 4.4.3.3. Создаваемая метапеременная, которая подставляется вместо предметной переменной x, имеет вид $?x\_n$, где n — целое положительное число (см. также замечание 3.2.4.1 на с. 79). Если будет найден унификатор, то такое именование метапеременных наглядно покажет, какое значение подходит для соответствующей предметной переменной. Также возможен программный доступ к переменной, вместо которой была подставлена метапеременная, с помощью метода `Metavariable.getBoundIndividualVariable()`.

Подформула (в коде — `qf`), корень дерева которой помечен квантором, вводимым данным правилом, имеет вид $\exists x A$ (или $\forall x A$). Для постро-

ения боковой формулы создается формула $A(mv|x)$ (обозначенная в коде `qsfWithSubstitution`) с помощью метода класса `Formula`

```
public Formula replaceVariable(BoundIndividualVariable x, Term t).
```

Этот метод создает новую формулу, отличающуюся от исходной только тем, что все экземпляры данной связанной переменной заменены на указанный терм.

Замечание 4.4.3.4. Поскольку переменная `x` представлена объектом класса `BoundIndividualVariable`, то для простоты реализации метода `replaceVariable()` желательно, чтобы в формуле A не было ни одной такой же переменной, но связанной вхождением квантора в A. Это действительно выполняется по определению чистой секвенции (см. определение 2.2.0.5 на с. 37, а также следующее за ним замечание).

Далее создается боковая формула `sideFormula` путем замены в главной формуле подформулы `qf` $(\exists x A)$ на `qsfWithSubstitution` $(A(mv|x))$ с помощью метода класса `Formula`

```
public Formula replaceSubformula(LogicalSymbolFormula subformula,
                                 Formula substitute).
```

Замечание 4.4.3.5. Напомним, что неатомарная подформула, являющаяся параметром `subformula` этого метода, однозначно идентифицируется в формуле, к которой применяется этот метод, в силу корректности разделения (см. раздел 4.2).

Наконец, построенная посылка становится непосредственным потомком узла заготовки в результате вызова метода `createChildren()`, который обсуждался в разделе 4.4.1.

4.4.4. Унификация и распознавание аксиом

В тот момент, когда тактика поиска вывода дает указание попытаться закрыть заготовку вывода, главный алгоритм поиска вывода вызывает

вспомогательный алгоритм унификации и либо продолжает наращивать заготовку, либо выдает ответ (см. раздел 3.2.4).

Унификация заготовки проводится с помощью метода

```
public Unification.Result unify(boolean oneUnifier)
```

класса `Unification`. Результат унификации, возвращаемый этим методом, является объектом класса `Unification.Result` с полями:

```
private Status status;
private java.util.List<InstantiationContext> unifiers
```

(и методами доступа к этим полям `getStatus()`, `getUnifiers()`). Поле `status` имеет своим значением одну из констант перечисления `Unification.Status`:

```
UNIFIABLE,
ESSENTIALLY_NONUNIFIABLE,
NONUNIFIABLE.
```

Эти константы означают, что заготовка унифицируема, существенно неунифицируема или неунифицируема соответственно. Поле `unifiers` является списком унификаторов. Если параметр `oneUnifier` метода `unify()` имеет значение `true`, то достаточно найти лишь один унификатор, в противном случае следует найти все унификаторы текущей заготовки.

Шаги алгоритма унификации описаны в разделе 3.2.2, поэтому здесь отметим лишь детали реализации, которые обеспечивают эффективность унификации.

При переборе значений метапеременных требуется подставлять термы вместо метапеременных в листовые секвенции заготовки вывода. Прямолинейная реализация такой подстановки заключалась бы в создании новых экземпляров секвенций и их формул-членов, в которых вместо метапеременных стояли бы соответствующие термы. При этом расходовалось бы много памяти. Предложен более экономный подход: новые экземпляры формул и секвенций не создаются, а каждая метапеременная принимает

значение-терм и ведет себя в некоторых отношениях как этот терм в тот промежуток времени, пока эта метапеременная имеет это значение. Опишем этот подход подробнее в терминах языка программирования Java.

В классе `Metavariable` определены следующие методы для работы со значением метапеременной:

```
public void setValue(Term t);
public void resetValue();
public Term getValue().
```

Первый метод устанавливает указанный терм в качестве значения метапеременной, второй — сбрасывает значение метапеременной, третий — выдает текущее значение метапеременной (или `null`, если значение не установлено). Взамен подстановки терма `t` вместо метапеременной в какую-либо формулу вызывается метод `setValue(t)`, устанавливающий значение этой метапеременной. В отношении равенства метапеременная ведет себя как ее текущее значение-терм: переопределенный в классе `Metavariable` метод

```
public boolean equals(Object obj),
```

сравнивающий объект, к которому применен этот метод, с объектом-параметром метода, возвращает такое же значение, как и метод `equals(obj)`, примененный к терму-значению метапеременной.

В соответствии в определением метода `equals()` переопределяется метод, который вычисляет хэш-код объекта,

```
public int hashCode()
```

так, что равные объекты имеют равные хэш-коды.

Замечание 4.4.4.1. Методы `equals()` и `hashCode()` определены в классе `Object` — корне иерархии всех классов языка программирования Java. В том же классе приведена общая спецификация этих методов. Другие классы при необходимости переопределяют эти методы, не нарушая общую спецификацию. (Подробнее см. [2, 65].)

Объекты, для которых методы `equals()` и `hashCode()` определены в соответствии с общей спецификацией этих методов, можно успешно хранить в хэш-таблицах общего назначения, например, `java.util.HashMap`. Класс `java.util.HashMap` реализует интерфейс `java.util.Map`, служащий для задания отображений. Покажем, как отображения используются при проверке секвенции на аксиоматичность.

При проверке секвенции на аксиоматичность строится система линейных неравенств, соответствующая этой секвенции. Такая система представляется объектом класса `AxiomRecognizer.LinearInequalitySystem` как список линейных неравенств — объектов класса `AxiomRecognizer.LinearInequality`. Линейное неравенство вида

$$a_1 \cdot x_1 + \ldots + a_n \cdot x_n + b < 0$$

или

$$a_1 \cdot x_1 + \ldots + a_n \cdot x_n + b \leq 0,$$

где $a_1, \ldots, a_n, b$ — рациональные числа, $x_1, \ldots, x_n$ — рациональнозначные переменные, удобно представлять тройкой, состоящей из

(1) отображения (экземпляра `java.util.Map`), сопоставляющего переменной x_i коэффициент a_i при этой переменной;

(2) свободного члена b этого неравенства;

(3) знака неравенства ($<$ или $\leq$).

Каждому предикатному символу со списком аргументов и каждой предикатной переменной, входящей в базовую подсеквенцию исследуемой на аксиоматичность секвенции, сопоставляется рациональнозначная переменная с помощью основанного на хэш-таблице отображения `java.util.HashMap`. (При этом не только у метапеременных, но и у термов, и у предикатных символов естественным образом переопределены методы `equals()` и `hashCode()`.)

Тогда неравенство, соответствующее КЦН, строится путем накопления коэффициентов этого строящегося неравенства при обходе синтаксического дерева этой КЦН в глубину.

Замечание 4.4.4.2. Не составило бы очень большого труда реализовать класс, представляющий отображение и специфичный для рассматриваемой задачи. Но желательно использовать уже написанный код (одной из основных целей объектно-ориентированного программирования является повторное использование кода). Таким образом, показано, как ценой гораздо меньших усилий использовать уже реализованный класс `java.util.HashMap`, и вдобавок существенно экономить память.

Наконец, каждая система линейных неравенств, получаемая при распознавании аксиом, проверяется на несовместность вспомогательным алгоритмом, описанным в главе 5, если все неравенства системы двучленные, иначе — функцией `FindInstance` системы компьютерной алгебры Mathematica [71] посредством J/Link — инструментария, связывающего программу на Java с Mathematica [67].

4.5. Предоставляемый программный интерфейс

Приведем систематический обзор составляющих предоставляемого программного интерфейса для поиска вывода. Мы приводим лишь обзор, а не полное описание, поскольку оно довольно объемно и может быть найдено в документации, сгенерированной утилитой `javadoc` [66] на основе исходного кода классов и включенных в код документирующих комментариев.

4.5.1. Создание формул и секвенций

Для создания экземпляров логических связок, предметных констант, предметных переменных, рациональных чисел, отрезков истинностных значений и формул (см. описание пакета `prover.calculus.syntax` в разделе 4.1.4) служат методы класса `LogicalFactoty`, среди которых:

```
LogicalConnective createConnective(java.lang.String s);
Moderator createModerator(RationalNumber q);
RationalNumber createRationalNumber(java.math.BigInteger numerator,
                                    java.math.BigInteger denominator);
QuantifierFormula createQuantifierFormula(Quantifier quantifier,
                                          BoundIndividualVariable variable,
                                          Formula subformula);
ConnectiveFormula createConnectiveFormula(LogicalConnective connective,
                                          java.util.List<Formula> operands);
TruthValueSegment createTruthValueSegment(RationalNumber start,
                                          RationalNumber end);
PredicateSymbol createPredicateSymbol(java.lang.String name,
                                      TruthValueSegment segment,
                                      java.util.List<Term> arguments);
IndividualConstant createIndividualConstant(java.lang.String name);
FreeIndividualVariable createFreeIndividualVariable(java.lang.String name);
BoundIndividualVariable createBoundIndividualVariable(java.lang.String name).
```

Экземпляр-одиночка класса `LogicalFactoty` возвращается методом `LogicalFactoty.getInstance()`.

Для создания объектов класса `Sequent` служат конструкторы этого класса, среди которых:

```
Sequent(java.util.List<Formula> listOfFormulas);
Sequent(boolean isTrusted, java.util.List<Formula> listOfFormulas).
```

Второй из этих конструкторов, если значение параметра `isTrusted` равно `false`, проверяет переданный список формул на чистоту и корректность разделения и генерирует исключение `java.lang.IllegalArgumentException`, если хотя бы одно из этих свойств не выполняется.

Также объект класса `Sequent` можно создать из строки, представляющей секвенцию, проведя синтаксический анализ этой строки с помощью класса `prover.Parser`. Объект этого класса создается конструктором, которому передается строка для анализа: `Parser(java.lang.String str)`. Синтаксический анализ и выдача секвенции производится методом `parse()`, который

- возвращает построенный объект класса `Sequent` или
- генерирует исключение `Parser.ParseException`, если анализируемая строка не соответствует форме Бэкуса-Наура для секвенций (см. раздел 4.3), или секвенция, представляемая этой строкой, не является чистой.

4.5.2. Поиск вывода секвенции

Перед тем, как начать поиск вывода некоторой секвенции, следует установить полный путь к ядру системы компьютерной алгебры Mathematica, вызвав статический метод

```
static void setMathematicaKernelPathName(java.lang.String path)
```

класса `Prover` и передав в качестве параметра строку, представляющую такой путь, например,

```
"C:\\Program Files\\Wolfram Research\\Mathematica\\5.2\\MathKernel.exe".
```

Если этот путь не установлен, то при распознавании аксиом системы линейных двучленных неравенств будут проверяться на несовместность с помощью алгоритма, описанного в главе 5, а при возникновении системы линейных неравенств с более чем двумя членами будет сгенерирована ошибка.

Далее нужно создать объект класса `Prover` с помощью одного из конструкторов:

```
Prover(Sequent sequent, Tactics tactics, ProofSearchOptions options);
Prover(Sequent sequent, Tactics tactics);
Prover(Sequent sequent, ProofSearchOptions options);
Prover(Sequent sequent).
```

Каждый из этих конструкторов принимает в качестве параметра секвенцию, вывод которой требуется найти. В первых двух конструкторах также задается тактика поиска вывода, а остальные конструкторы строят объект, который будет использовать тактику поиска вывода по умолчанию — `DefaultTactics`. В первом и третьем конструкторах задаются опции поиска вывода с помощью объекта класса `ProofSearchOptions`, остальные

конструкторы строят объект, который будет руководствоваться опциями по умолчанию.

Опции поиска вывода

Опции поиска вывода задаются объектом класса `ProofSearchOptions`. Каждая опция имеет ключ и значение. Ключ — это одна из констант перечисления `ProofSearchOptions.OptionKey`. Значение — объект, точный тип которого зависит от опции и специфицирован ниже.

С опциями можно работать с помощью следующих методов класса `ProofSearchOptions`. Задать опцию можно с помощью метода

```
void put(OptionKey key, Object value).
```

Для получения опции, соответствующей указанному ключу, служит метод

```
Object get(OptionKey key).
```

Удалить опцию можно с помощью метода

```
void remove(OptionKey key).
```

Проверить наличие хотя бы одной заданной опции можно с помощью метода

```
boolean isEmpty().
```

Для получения множества пар, состоящих из ключей и значений опций, служит метод

```
java.util.Set<java.util.Map.Entry<OptionKey, Object>> pairs().
```

Поддерживаемые опции приведены в следующей таблице, там же указано значение каждой опции по умолчанию.

Ключ	Тип	Значение по умолчанию	Назначение опции
`TIMEOUT_KEY`	`Long`	0	Лимит времени в миллисекундах (0 означает бесконечность), отведенный на поиск вывода; если этот лимит исчерпан, то поиск вывода должен завершиться.
`ONE_UNIFIER_KEY`	`Boolean`	`false`	Если `true`, то при унификации достаточно найти только один унификатор (если он существует), иначе — все унификаторы.
`SAVE_MEMORY_KEY`	`Boolean`	`true`	Если `true`, то в заготовке вывода хранятся только листья, иначе — все промежуточные узлы тоже хранятся.

Тактики поиска вывода

Любая тактика поиска вывода реализует интерфейс `Tactics`. Этот интерфейс имеет три метода. Метод, определяющий очередной шаг главного алгоритма поиска вывода,

```
Tactics.NextStep chooseNextStep()
```

детально обсуждался в разделе 4.4.2, как и выдаваемый этим методом ответ. Остальные два метода

```
void setSkeleton(ProofSkeleton skeleton);
ProofSkeleton getSkeleton()
```

служат для задания (до начала поиска вывода) и получения заготовки вывода, исследуемой данной тактикой, соответственно.

Запуск поиска вывода

Запуск поиска вывода производится методом `prove()` класса `Prover`. Этот метод возвращает результат поиска вывода — одну из констант перечисления `Prover.ProofStatus`:

`PROVABLE`, если исходная секвенция выведена;

`UNPROVABLE`, если установлено, что исходная секвенция невыводима;

`UNKNOWN`, если не удалось получить ни один из двух предыдущих ответов, а отведенное время истекло, или доступная данной программе память исчерпана.

4.5.3. Получение информации о ходе поиска вывода

Разработанный ИПП предоставляет возможность получать информацию о текущем шаге поиска вывода. Для задания получателя такой информации служит интерфейс `ProofListener`. Информация о текущем шаге поиска вывода представляется объектом класса `ProofEvent`. Будем называть такой объект событием, а объект какого-либо класса, реализующего интерфейс `ProofListener`, слушателем событий. Получение события заключается в вызове соответствующего этому событию метода слушателя и передаче этого события в качестве параметра этого метода.

Чтобы слушатель `listener` получал события о ходе поиска вывода, осуществляемого объектом `prover` класса `Prover`, нужно зарегистрировать этого слушателя с помощью метода

```
void addProofListener(ProofListener listener)
```

класса `Prover` таким образом: `prover.addProofListener(listener)`.

Для удаления слушателя и получения всех слушателей, зарегистрированных у какого-либо объекта класса `Prover`, служат следующие методы этого класса:

`void removeProofListener(ProofListener listener);`
`ProofListener[] getProofListeners()`.

Опишем методы интерфейса `ProofListener`, моменты, когда эти методы вызываются, и информацию, содержащуюся в передаваемых этим методам событиях.

1. Метод

```
void proofSearchStarted(ProofEvent event)
```

вызывается в самом начале поиска вывода. Следующие методы класса `ProofEvent`

```
Prover getProver();
ProofSkeleton getSkeleton();
Tactics getTactics();
```

вызванные на переданном событии, возвращают используемые объекты, представляющие главный алгоритм поиска вывода, заготовку вывода и тактику соответственно. (Остальные методы `ProofEvent` возвращают `null`.)

2. Метод

```
void ruleApplied(ProofEvent event)
```

вызывается сразу после каждого контрприменения правила вывода. Вдобавок к методам класса `ProofEvent`, перечисленным в п. 1, метод

```
Tactics.RuleChoice getRuleChoice()
```

возвращает объект, представляющий выбор тактики поиска вывода (в соответствии с этим выбором только что было произведено контрприменение правила).

3. Метод

```
void closingSkeleton(ProofEvent event)
```

вызывается непосредственно перед попыткой закрытия текущей заготовки вывода. Информацию дают методы класса `ProofEvent`, которые перечислены в п. 1.

4. Метод

```
void metavariablesInstantiated(ProofEvent event)
```

вызывается сразу после каждой конкретизации всех метапеременных в текущей заготовке при унификации. Информацию дают методы класса `ProofEvent`, которые перечислены в п. 1.

5. Метод

```
void systemOfInequalitiesConstructed(ProofEvent event)
```

вызывается сразу после каждого построения системы линейных неравенств при унификации. Информацию дают методы класса `ProofEvent`, которые перечислены в п. 1, а также

```
Sequent getSequent();
AxiomRecognizer.LinearInequalitySystem getInequalitySystem().
```

Последние два метода выдают секвенцию и систему линейных неравенств, соответствующую этой секвенции.

6. Метод

```
void sequentTestedForAxiom(ProofEvent event)
```

вызывается сразу после завершения каждой проверки секвенции на аксиоматичность при унификации. Информацию дают методы класса `ProofEvent`, которые перечислены в п. 1, а также

```
Sequent getSequent();
java.lang.Boolean isSequentAxiom().
```

Последние два метода выдают соответственно секвенцию и булевское значение, говорящее о том, является ли аксиомой эта секвенция.

7. Метод

`void proofSearchFinished(ProofEvent event)`

вызывается при завершении поиска вывода. Информацию дают методы класса `ProofEvent`, которые перечислены в п. 1, а также

`Prover.ProofStatus getProofStatus();`
`java.util.List<InstantiationContext> getUnifiers().`

Предпоследний метод выдает результат поиска вывода. Последний метод выдает список унификаторов, если они найдены (и `null` в противном случае).

Пример использования разработанного ИПП для поиска вывода содержится в приложении А.

Глава 5.

Алгоритм решения систем линейных двучленных неравенств и его программная реализация

В этой главе приводятся алгоритмы проверки совместности и решения систем линейных двучленных неравенств. Доказывается корректность этих алгоритмов. Даются оценки их временной сложности, что позволяет сделать вывод об их сильной полиномиальности. Описывается разработанный интерфейс прикладного программирования для решения систем указанного вида. Приводятся полученные экспериментальные результаты, говорящие о большей эффективности программной реализации этих алгоритмов по сравнению с системой компьютерной алгебры Mathematica.

5.1. Алгоритм проверки совместности систем

Рассмотрим задачу проверки совместности системы из m строгих и нестрогих линейных двучленных неравенств с целочисленными коэффи-

циентами относительно n рациональнозначных переменных.

В разделе 1.1.4 было дано определение сильно полиномиального алгоритма, и упомянут алгоритм проверки совместности систем из [18]. Ниже приводится уточненное описание этого алгоритма и оценка его сложности в формальной вычислительной модели.

5.1.1. Постановка задачи и основные определения

Определение 5.1.1.1. Переменные, входящие в исходную систему неравенств, будем называть *основными*. Для удобства изложения наряду с основными переменными, введем *дополнительные переменные* $\mathbf{0}$ (*нулевая переменная*) и $\mathbf{1}$ (*единичная переменная*) для чисел 0 и 1 соответственно. Далее под переменными будем понимать как основные, так и дополнительные переменные.

Определение 5.1.1.2. Под *ограничением* будем понимать линейное неравенство вида $ax + by \prec 0$, где $\prec$ — знак $<$ или $\leq$, a и b — целые числа — коэффициенты, x и y — переменные (основные или дополнительные).

Пример 5.1.1.3. Двучленные неравенства $2x + 3 < 0$ и $4x \leq 0$ представляются в виде ограничений как $2x + 3 \cdot \mathbf{1} < 0$ и $4x + 0 \cdot \mathbf{0} \leq 0$ соответственно.

Определение 5.1.1.4. Два ограничения $C_1 = a_1x_1 + b_1y_1 \prec_1 0$ и $C_2 = a_2x_2 + b_2y_2 \prec_2 0$ будем считать *равными*, если

(1) $\prec_1 = \prec_2$; и

(2) верно хотя бы одно из двух:

(2a) $x_1 = x_2$, $a_1 = a_2$, $y_1 = y_2$, и $b_1 = b_2$; или

(2b) $x_1 = y_2$, $a_1 = b_2$, $y_1 = x_2$, и $b_1 = a_2$.

Определение 5.1.1.5. *Системой* назовем конъюнкцию конечного числа ограничений.

Требуется определить, существует ли вектор рациональных значений основных переменных, удовлетворяющий каждое ограничение системы, т.е. требуется определить, совместна ли система. Иногда требуется решить систему, т.е. в дополнение к предыдущей задаче требуется найти хотя бы один удовлетворяющий систему вектор в случае совместности системы.

Определение 5.1.1.6. Ограничение $ax + by \prec 0$ назовем *приведенным*, если

(1) обе переменные x и y являются переменной $\mathbf{0}$, или эти переменные не совпадают; и

(2) если ровно одна из этих переменных является переменной $\mathbf{0}$, то абсолютная величина коэффициента при другой переменной равна 1; и

(3) если какая-либо из этих переменных — $\mathbf{0}$, то коэффициент при этой переменной — 0.

Определение 5.1.1.7. Пусть дано ограничение C. Назовем *приведением* C и обозначим $red(C)$

- само ограничение C, если его переменные не совпадают, и ни одна его переменная не является $\mathbf{0}$;
- (единственное) приведенное ограничение, равносильное ограничению C, в противном случае.

Определение 5.1.1.8. Ограничение $ax + by \prec 0$ назовем *правильным*, если оно приведенное, и переменные x и y не являются $\mathbf{0}$ одновременно.

Обозначение 5.1.1.9. Обозначим $S^{\{x,y\}}$ множество ограничений системы S, в которые одновременно входят переменные x и y.

Зафиксируем константу $r_0 = 6$ (ее выбор объясняется нижеследующим описанием алгоритма; на самом деле r_0 можно положить равным любому достаточно большому числу, при этом асимптотические оценки сложности описываемого алгоритма не изменятся).

Определение 5.1.1.10. Будем называть систему S *правильной*, если все ограничения в ней правильны, и для любых двух переменных x, y выполняется $|S^{\{x,y\}}| \leq r_0$, где $|Set|$ — мощность множества Set.

Определение 5.1.1.11. Пусть правильные ограничения $C_1 = a_1x + b_1y \prec_1 0$ и $C_2 = a_2y + b_2z \prec_2 0$ таковы, что $b_1a_2 < 0$. Тогда определим *композицию* этих ограничений относительно переменной y как $C_1 \circ_y C_2 = a_1|a_2|x + b_2|b_1|z \prec 0$, где $\prec$ — знак $\leq$, если оба $\prec_1$ и $\prec_2$ — нестрогие, и $<$ иначе, $|a|$ — абсолютная величина числа a.

Легко проверить, что композиция ограничений ассоциативна, т.е. ограничения $(C_1 \circ_{y'} C_2) \circ_{y''} C_3$ и $C_1 \circ_{y'} (C_2 \circ_{y''} C_3)$ определены одновременно и совпадают.

Алгоритм проверки совместности системы заключается в последовательном исключении переменных из системы и одновременном с исключением переменной удалении избыточных ограничений так, что количество ограничений, содержащих любые две переменные, не превосходит константу r_0.

Сначала приведем описание вспомогательных алгоритмов.

5.1.2. Вспомогательный алгоритм проверки совместности систем обычным методом исключений переменных

Опишем алгоритм $IsConsistent(T)$, проверяющий совместность непустой системы T. Этот алгоритм представляет собой детализированный метод исключений переменных (ср. [46, раздел 12.2]).

(1) T_0 полагается равным пустой системе.
Для каждого ограничения C системы T

(1.1) если $red(C)$ содержит только дополнительные переменные, и $red(C)$ не выполняется при замене этих дополнительных пере-

менных на соответствующие им числа, то выдается ответ «несовместна»;

(1.2) если $red(C)$ содержит хотя бы одну основную переменную, то $red(C)$ добавляется в T_0.

(2) Для каждого $i = 1, \ldots, n$ производится исключение одной основной переменной следующим образом.

(2.1) Если система T_{i-1} пуста, то выдается ответ «совместна».

(2.2) Выбирается любая основная переменная y системы T_{i-1}, и строится система T_i.

(2.3) В систему T_i переносятся все ограничения системы T_{i-1}, в которые не входит переменная y.

(2.4) Для всех пар ограничений C_1 и C_2 системы T_{i-1}, содержащих переменную y, если определена композиция $C = C_1 \circ_y C_2$, то вычисляется $red(C)$.

(2.4.1) Если $red(C)$ содержит только дополнительные переменные, и $red(C)$ не выполняется при замене этих дополнительных переменных на соответствующие им числа, то выдается ответ «несовместна».

(2.4.2) Если $red(C)$ содержит хотя бы одну основную переменную, то $red(C)$ добавляется в T_i.

(3) Если система T_n пуста, то выдается ответ «совместна».

(4) Если система T_n (она содержит лишь дополнительные переменные) выполнена при замене $\mathbf{0}$ на 0 и $\mathbf{1}$ на 1, то выдается ответ «совместна», иначе — «несовместна».

Предложение 5.1.2.1. *Пусть дана непустая система T. Тогда алгоритм $IsConsistent(T)$ выдает ответ «несовместна», если T несовместна, и выдает ответ «совместна», если T совместна.*

Д о к а з а т е л ь с т в о. Ясно, что алгоритм всегда заканчивает свою работу за конечное число шагов, и что на каждом из шагов по исключению переменной строятся системы T_i $(i = 1, 2, \ldots, n)$, которые совместны или несовместны одновременно с исходной системой T (см. также раздел 12.2 в [46]). Поэтому ответ, выдаваемый алгоритмом на шаге (4), верен.

Пустота системы T_i $(i = 1, 2, \ldots, n)$ означает, что исключаемая из системы T_{i-1} переменная y входит во все ограничения системы T_{i-1}, и для любых значений остальных переменных найдется значение переменной y, при котором все ограничения системы T_{i-1} выполняются. Поэтому ответ, выдаваемый алгоритмом в случае обнаружения пустоты системы (на шаге (2.1) или (3)), верен.

Наконец, ответ, выдаваемый на шаге (1.1) или (2.4.1), верен, поскольку построенное ограничение $red(C)$ является следствием исходной системы. □

5.1.3. Вспомогательный алгоритм добавления ограничения в правильную систему

Считаем, что переменные представлены словами из букв английского алфавита и десятичных цифр, причем такие слова начинаются с буквы. Символы упорядочены:
$0 < 1 < \ldots < 9 < A < B \ldots < Z < a < b < \ldots < z$.

Обозначение 5.1.3.1. Пусть $\prec_V$ — отношение лексикографического порядка на множестве переменных.

Определение 5.1.3.2. Зададим линейный квазипорядок (см., например, [7, с. 36]) на множестве правильных ограничений, каждое из которых содержит одни и те же две переменные x и y, $x \prec_V y$, и знак коэффициентов при x в каждом из ограничений этого множества одинаков: ограничение $C_1 = a_1x + b_1y \prec_1 0$ *предшествует* ограничению $C_2 = a_2x + b_2y \prec_2 0$, если $b_1a_2 \leq a_1b_2$.

Опишем вспомогательный алгоритм $FindRedundantInequality$, на вход которому подается упорядоченная (в соответствии с определением 5.1.3.2) последовательность из 3 правильных ограничений $C_i = a_i x + b_i y \prec_i 0$ $(i = 1, 2, 3)$.[1]

(1) Если знак $\prec_2$ есть знак $\leq$, или все $\prec_i$ $(i = 1, 2, 3)$ суть $<$, то выдается ответ C_2.

(2) Если $\prec_1$ есть знак $\leq$, $\prec_2$ — $<$, $\prec_3$ — $<$, то при $b_1 a_2 = a_1 b_2$ выдается C_1, а при $b_1 a_2 < a_1 b_2$ выдается C_2.

(3) Если знаки $\prec_i$ расположены в таком порядке: $<<\leq$, то при $b_2 a_3 = a_2 b_3$ выдается C_3, а при $b_2 a_3 < a_2 b_3$ выдается C_2.

(4) Если же порядок знаков $\prec_i$ таков: $\leq<\leq$, то выдается ответ «нет лишнего ограничения».

Нетрудно убедиться, что алгоритм $FindRedundantInequality$, если выдает ограничение, то оно является лишним из 3 ограничений, поданных ему на вход (лишнее ограничение — то, удаление которого не изменяет множество решений системы из данных 3 ограничений). Таким образом, имеет место

Предложение 5.1.3.3. *Пусть алгоритму $FindRedundantInequality$ на вход подана упорядоченная последовательность из правильных ограничений C_1, C_2, C_3. Тогда если этот алгоритм выдал в качестве ответа ограничение C_k, то $(C_1 \& C_2 \& C_3 \iff C_i \& C_j)$, где i, j, k — перестановка чисел $1, 2, 3$; а если этот алгоритм выдал ответ «нет лишнего ограничения», то C_1 имеет знак $\leq$, C_2 — $<$, C_3 — $\leq$.*

Опишем вспомогательный алгоритм $AddConstraint(R, C)$, который служит для добавления ограничения $C = ax + by \prec 0$ в правильную систему R.

[1] Имеем $b_1 a_2 \leq a_1 b_2$ и $b_2 a_3 \leq a_2 b_3$, что при положительных a_i эквивалентно $b_1/a_1 \leq b_2/a_2 \leq b_3/a_3$.

(1) Если $red(C)$ содержит только дополнительные переменные, то

(1.1) если $red(C)$ не выполняется при замене этих дополнительных переменных на соответствующие им числа, то выдается ответ «несовместна»;

(1.2) если же $red(C)$ выполняется, то система R выдается в качестве ответа.

(2) Иначе, если $red(C)$ содержит ровно одну ненулевую переменную x, то $red(C)$ добавляется в множество $R^{\{\mathbf{0},x\}}$, и полученная в результате этого добавления система выдается как ответ.

(3) Иначе (red(C) = $ax + by \prec 0$ содержит две ненулевые переменные x и y), строится система T, состоящей из ограничений $R^{\{\mathbf{0},x\}}$, $R^{\{\mathbf{0},y\}}$, $R^{\{x,y\}}$ и $-ax - by \prec' 0$, где $\prec'$ — знак $<$, если $\prec$ — знак $\leq$, и наоборот. Если алгоритм $IsConsistent(T)$ выдал, что система T несовместна, то система R выдается как ответ, иначе производятся следующие действия.

(3.1) $red(C)$ добавляется в множество $R^{\{x,y\}}$.

(3.2) Выбирается меньшая (по отношению порядка $\prec_V$ на множестве переменных) из переменных x и y, для определенности x. Находится знак $\oplus$ (либо $+$, либо $-$) коэффициента при x в $red(C)$. Из $R^{\{x,y\}}$ выделяется подмножество $R^{\{x,y\}}_{\oplus x}$ ограничений с коэффициентами знака $\oplus$ при x. Если $|R^{\{x,y\}}_{\oplus x}| < 3$, то система, полученная из R добавлением $red(C)$ на шаге (3.1), выдается как ответ.

(3.3) Ограничения из $R^{\{x,y\}}_{\oplus x}$ упорядочиваются (в соответствии с определением 5.1.3.2).

(3.4) Если упорядоченная последовательность ограничений, принадлежащих $R^{\{x,y\}}_{\oplus x}$, состоит из 3 ограничений C_1, C_2, C_3, то вызывается $FindRedundantInequality(C_1, C_2, C_3)$. Если $FindRedundantInequality$ выдал ограничение, то оно удаляется из $R^{\{x,y\}}_{\oplus x}$, и полученная система выдается как ответ.

(3.5) Если упорядоченная последовательность ограничений, принадлежащих $R^{\{x,y\}}_{\oplus x}$, состоит из 4 ограничений $C_i = a_i x + b_i y \prec_i 0 \quad (i = 1, 2, 3, 4)$, то вызывается $FindRedundantInequality(C_1, C_2, C_3)$.

(3.5.1) Если $FindRedundantInequality(C_1, C_2, C_3)$ выдал ограничение C_k, то оно удаляется из $R^{\{x,y\}}_{\oplus x}$.
Если $FindRedundantInequality(C_i, C_j, C_4)$ (где C_i, C_j, C_4 — последовательность, полученная из C_1, C_2, C_3, C_4 вычеркиванием C_k) выдал ограничение, то оно удаляется из $R^{\{x,y\}}_{\oplus x}$, и полученная система выдается как ответ.

(3.5.2) Иначе (т.е. $FindRedundantInequality(C_1, C_2, C_3)$ выдал ответ «нет лишнего ограничения») вызывается $FindRedundantInequality(C_2, C_3, C_4)$.

(3.5.2.1) Если $FindRedundantInequality(C_2, C_3, C_4)$ выдал ограничение C_k, то оно удаляется из $R^{\{x,y\}}_{\oplus x}$.

(3.5.2.2) Если $FindRedundantInequality(C_1, C_i, C_j)$ (где C_1, C_i, C_j — последовательность, полученная из C_1, C_2, C_3, C_4 вычеркиванием C_k) выдал ограничение, то оно удаляется из $R^{\{x,y\}}_{\oplus x}$, и полученная система выдается как ответ.

(3.6) В остальных случаях (отличных от рассмотренных в (3.4) и (3.5)) система, полученная из R добавлением $red(C)$ на шаге (3.1), выдается как ответ.

Предложение 5.1.3.4. *Пусть система R получена путем добавления ограничений в изначально пустую систему с помощью алгоритма AddConstraint; ограничение C есть $ax + by \prec 0$, где $x \prec_V y$. Тогда в результате вызова $AddConstraint(R, C)$ либо выдается ответ «несовместна», либо выдается система, причем если получен ответ «несовместна», то система $R\&C$ несовместна; а если получена система R', то*

(a) R' *правильна и равносильна системе* $R\&C$*; и*

(b) R' *совпадает с* R*, если* x *и* y *различные ненулевые переменные, и* $R^{\{\mathbf{0},x\}}\&R^{\{\mathbf{0},y\}}\&R^{\{x,y\}} \Rightarrow C$.

Доказательство. Ответ «несовместна» выдается в том и только в том случае, когда ограничение C содержит только дополнительные переменные (т.е. является константным) и не выполняется; поэтому система $R\&C$ несовместна.

Легко видеть, что выполнение условия $R^{\{\mathbf{0},x\}}\&R^{\{\mathbf{0},y\}}\&R^{\{x,y\}} \Rightarrow red(C)$ эквивалентно несовместности системы T, которая строится на шаге (3). Если система T несовместна, то исходная система выдается в качестве ответа; таким образом, свойство (b) установлено.

При построении системы, являющейся ответом, удаляются только лишние ограничения в соответствии с указанием алгоритма $FindRedundantInequality$. Поэтому полученная система эквивалентна системе $R\&C$ в силу предложения 5.1.3.3.

Докажем правильность полученной системы R'. $|R^{\{\mathbf{0},x\}}| < r_0$ (см. шаг (2)) в силу требования (2) определения 5.1.1.6.

Теперь для установления правильности системы R' достаточно показать, что добавление ограничения с ненулевыми переменными (см. шаг (3) и его подшаги) сохраняет следующее свойство: $|R'^{\{x,y\}}_{\oplus x}| < 3$, или $|R'^{\{x,y\}}_{\oplus x}| = 3$ и при упорядочении 3 ограничений из $R'^{\{x,y\}}_{\oplus x}$ их знаки располагаются таким образом: $\leq<\leq$.

Когда после добавления $red(C)$ $|R^{\{x,y\}}_{\oplus x}|$ становится равной 3 (см. шаг (3.4)), производится удаление одного лишнего ограничения во всех случаях, кроме случая такого порядка знаков упорядоченных ограничений из $R^{\{x,y\}}_{\oplus x}$: $\leq<\leq$. Если в $R^{\{x,y\}}_{\oplus x}$ добавляется четвертое ограничение, то все случаи взаимного расположения знаков этих четырех ограничений C_1, C_2, C_3, C_4 таковы: $<\leq<\leq$, $\leq<<\leq$, $\leq<\leq<$, $\leq\leq<\leq$, $\leq<\leq\leq$. Следовательно, из C_1, C_2, C_3 или из C_2, C_3, C_4 можно удалить лишнее ограничение. Это и делается на шагах (3.5.1) и (3.5.2), причем попытка удаления лишнего ограничения повторяется, чтобы в $R'^{\{x,y\}}_{\oplus x}$ оставались 3 ограничения

только в одном случае расположения знаков: $\leq<\leq$. $\square$

5.1.4. Главный алгоритм проверки совместности систем

Опишем главный алгоритм $IsConsistentWithReduction(S)$, определяющий, совместна ли непустая система S.

(1) S_0 полагается равной пустой системе.
Для каждого ограничения C системы S

(1.1) вызывается алгоритм $AddConstraint(S_0, C)$;

(1.2) если $AddConstraint(S_0, C)$ вернул ответ «несовместна», то выдается ответ «несовместна»;

(1.3) иначе система S_0 полагается равной полученному ответу $AddConstraint(S_0, C)$.

(2) V_0 полагается равным множеству всех переменных системы S_0, кроме нулевой переменной.
Для каждого $i = 1, \ldots, n$ производится исключение основной переменной следующим образом.

(2.1) Если система S_{i-1} пуста, то выдается ответ «совместна».

(2.2) Выбирается любая основная переменная y системы S_{i-1}, полагается $V_i = V_{i-1} \setminus \{y\}$, и строится система S_i.

(2.3) В систему S_i переносятся все ограничения системы S_{i-1}, в которые не входит переменная y.

(2.4) Для всех ограничений $C_1 \in S_{i-1}^{\{\mathbf{0},y\}}$ и всех ограничений $C_2 \in (\cup_{x \in V_i} S_{i-1}^{\{x,y\}}) \cup S_{i-1}^{\{\mathbf{0},y\}}$, если определена композиция $C_1 \circ_y C_2$, добавляем эту композицию в S_i с помощью вызова $AddConstraint(S_i,\ C_1 \circ_y C_2)$.
При этом если $AddConstraint$ вернул ответ «несовместна», то выдается ответ «несовместна», иначе S_i полагается равной полученному ответу $AddConstraint$.

(2.5) Для всех пар ограничений $C_1, C_2 \in \cup_{x \in V_i} S_{i-1}^{\{x,y\}}$, если определена композиция $C_1 \circ_y C_2$, добавляем эту композицию в S_i с помощью вызова $AddConstraint(S_i,\ C_1 \circ_y C_2)$.
При этом если $AddConstraint$ вернул ответ «несовместна», то выдается ответ «несовместна», иначе S_i полагается равной полученному ответу $AddConstraint$.

(3) Если система S_n пуста, то выдается ответ «совместна».

(4) Если система S_n (она содержит лишь дополнительные переменные) выполнена при замене **0** на 0 и **1** на 1, то выдается ответ «совместна», иначе — «несовместна».

Корректность алгоритма $IsConsistentWithReduction$ обосновывается аналогично корректности алгоритма $IsConsistent$ (см. предложение 5.1.2.1) с учетом предложения 5.1.3.4. Таким образом, имеет место

Теорема 5.1.4.1. *Пусть дана непустая система S. Тогда алгоритм $IsConsistentWithReduction(S)$ выдает ответ «несовместна», если S несовместна, и выдает ответ «совместна», если S совместна.*

Замечание 5.1.4.2. Общая схема главного алгоритма и идея нахождения лишнего ограничения заимствованы из [18]. Детализация этого алгоритма, особенно в части удаления лишних ограничений, проведена автором диссертации, также как описываемые ниже оценка сложности и программная реализация алгоритма.

5.2. Оценка временной сложности алгоритма проверки совместности систем

5.2.1. Вычислительная модель

Определим вычислительную модель (см. главу 1 в [4]), в которой будет оценена временная сложность алгоритма $IsConsistentWithReduction$

проверки совместности систем.

Будем использовать *равнодоступную адресную машину* (см. разделы 1.2, 1.3 в [4]) с *логарифмическим весовым критерием*, модифицированным следующим образом. Учитывается ограниченность реальной ячейки памяти, так что аддитивные операции над двумя целыми числами длиной не более l каждое выполняются за время $O(l)$ (вне зависимости от способа адресации операндов). В [4] логарифмический весовой критерий также дает линейное время для мультипликативных операций, что нереалистично, поскольку неизвестны алгоритмы умножения длинных целых чисел за линейное время. Более реалистичным представляется выполнение мультипликативных операций над двумя целыми числами длиной не более l каждое за время $O(l^2)$ (также вне зависимости от способа адресации операндов). Хотя для умножения длинных целых чисел существуют более быстрые алгоритмы, чем квадратичный, будем использовать квадратичную оценку, поскольку этого достаточно для получения верхней оценки временной сложности.

5.2.2. Представление системы в памяти

В алгоритме $IsConsistentWithReduction$ часто выполняется операция выбора всех ограничений, содержащих две данные переменные. Для эффективной реализации этой операции введем строгий линейный порядок на множестве неупорядоченных пар переменных, и будем использовать сбалансированное дерево поиска для хранения ограничений.

Определение 5.2.2.1. Неупорядоченная пара переменных $\{u_1, u_2\}$ *предшествует* паре $\{w_1, w_2\}$, если эти пары не равны, и соответствующая упорядоченная пара (u_{i_1}, u_{i_2}), где $u_{i_1} \prec_V u_{i_2}$, лексикографически предшествует соответствующей упорядоченной паре (w_{j_1}, w_{j_2}), где $w_{j_1} \prec_V w_{j_2}$. (Напомним, что $\prec_V$ — линейный порядок на множестве переменных, см. обозначение 5.1.3.1 на с. 132.)

Каждая система S_i $(i = 0, 1, \ldots, n)$, которую строит алгоритм

$IsConsistentWithReduction$, в памяти представляется с помощью сбалансированного дерева поиска, ключом которого является пара переменных $\{x, y\}$, а значением, соответствующим ключу, — множество $S_i^{\{x,y\}}$. Такое представление гарантирует логарифмическое по числу узлов дерева время поиска ключа.

Кодирование переменных естественно считать сжатым, не избыточным (см. раздел 2.1 в [17]), например, переменные представляются как $x1, \ldots, xn$.

5.2.3. Оценка временной сложности

Теорема 5.2.3.1. *Для каждого $i = 0, 1, \ldots, n$ любое ограничение $C = ax + by \prec 0$ системы S_i, где x, y — ненулевые переменные, представляется в виде композиции различных ограничений системы S_0: $C = C_{j_1} \circ_{x_{j_1}} C_{j_2} \circ_{x_{j_2}} \ldots \circ_{x_{j_{k-1}}} C_{j_k}$.*

Доказательство полностью аналогично доказательству леммы 9 в [18]. Изложим схему доказательства. Непосредственно из построения алгоритма $IsConsistentWithReduction$ видно, что любое ограничение системы S_i, в котором нет нулевой переменной, представляется в виде композиции ограничений системы S_0. Менее тривиально устанавливается, что ограничения системы S_0, композиция которых является не содержащим нулевой переменной ограничением системы S_i, различны. Доказательство этого утверждения проводится с помощью метода возвратной математической индукции по i. Индукционный переход осуществляется с помощью индукции по количеству вызовов алгоритма $AddConstraint$ при исполнении шага (2.5) алгоритма $IsConsistentWithReduction$ и опирается на то, что если построена композиция двух ограничений, в представлении каждого из которых есть одно и то же ограничение системы S_0, то алгоритм $AddConstraint$ не добавит такое ограничение в систему в соответствии со свойством (b) этого алгоритма (см. предложение 5.1.3.4). □

Из теоремы 5.2.3.1 следует, что для каждого $i = 0, 1, \ldots, n$ дли-

на любого коэффициента системы S_i, которая строится алгоритмом $IsConsistentWithReduction$, не превосходит сумму длин всех коэффициентов системы S_0. Построение правильной системы S_0 из исходной системы S, очевидно, не удлиняет коэффициенты и не увеличивает количество ограничений. Таким образом, длина любого коэффициента системы S_i не превосходит L, где L — сумма длин всех коэффициентов исходной системы S. Одна композиция двух ограничений и несколько ограничений с такими коэффициентами могут быть поданы на вход вспомогательному алгоритму $IsConsistent$, осуществляющему исключение не более двух переменных. Поэтому при работе $IsConsistent$ могут появиться числа длиной вплоть до $6L$. Поскольку в алгоритме $IsConsistentWithReduction$ отсутствуют операции, которые удлиняют числа более значительно, то в целом алгоритм выполняет арифметические операции над числами длиной не более $6L$.

Если каждое число из исходной системы занимает не более k двоичных разрядов, то $L \leq 2km$, и длина чисел, используемых в работе алгоритма, не превосходит $12km$.

Далее для оценки временной сложности алгоритма $IsConsistentWithReduction$ достаточно того, что этот алгоритм выполняет арифметические операции над числами длиной $O(L)$, где L — сумма длин всех коэффициентов исходной системы.

Вспомогательный алгоритм $IsConsistent$ используется только для проверки совместности систем, содержащих не более двух основных переменных. Алгоритм $IsConsistent$ осуществляет обычное исключение не более двух переменных из системы с $O(1)$ ограничениями, коэффициенты которых имеют длину $O(L)$; при этом наиболее трудоемкая операция — вычисление произведений таких коэффициентов при построении композиций. Потому алгоритм $IsConsistent$ работает за время $O(L^2)$.

Для системы S_i $(i = 0, 1, \ldots, n)$ с не более чем n основными переменными алгоритм $AddConstraint$ находит приведение добавляемого ограничения за время $O(L)$; затем выполняет поиск трех пар переменных в сбалан-

сированном дереве с $O(n^2)$ узлами, на что затрачивает время $O(\log n)$. Далее обращается к алгоритму $IsConsistent$, затрачивая на это время $O(L^2)$. Наконец, $AddConstraint$ осуществляет удаление одного или двух избыточных ограничений из множества ограничений, мощность которого не превосходит r_0, на что уходит время $O(L^2)$. Таким образом, $AddConstraint$ работает за время $O(\log n + L^2)$.

Алгоритм $IsConsistentWithReduction$ при построении правильной системы S_0 (см. описание алгоритма $IsConsistentWithReduction$, шаг (1)) вызывает $AddConstraint$ m раз, затрачивая время $O(m(\log n + L^2))$.

Далее на каждом из не более чем n шагов по исключению основной переменной из системы S_i (см. описание алгоритма $IsConsistentWithReduction$, шаг (2)) строятся композиции пар ограничений, в которых фиксирована одна (исключаемая) переменная. В силу правильности системы S_i таких ограничений $O(n-i) \cdot r_0 = O(n)$, а таких пар ограничений — $O(n^2)$. Композиция находится за время $O(L^2)$. Для каждой из $O(n^2)$ композиций ограничений вызывается $AddConstraint$. Значит, исключение всех основных переменных требует времени $O(n^3(\log n + L^2))$.

На последнем шаге алгоритма $IsConsistentWithReduction$ проверяется выполнение правильной системы S_n с одними лишь дополнительными переменными, в такой системе не более $O(1)$ ограничений. На эту проверку уходит время $O(L)$.

Итак, в заданной вычислительной модели алгоритм $IsConsistentWithReduction$ имеет временную сложность $O(m \log n + mL^2 + n^3 \log n + n^3 L^2)$, где m — число неравенств, n — число переменных, L — сумма длин всех коэффициентов исходной системы.

Замечание 5.2.3.2. Полученная оценка временной сложности показывает, что алгоритм $IsConsistentWithReduction$ полиномиален на модифицированной равнодоступной адресной машине (эта машина описана в разделе 5.2.1). Легко показать, что этот алгоритм полиномиален и на равнодо-

ступной адресной машине (см. [4, разделы 1.2, 1.3]), следовательно, по теореме 1.4 из [4] данный алгоритм полиномиален на многоленточной машине Тьюринга. Учитывая, что алгоритм проверки совместности систем требует $O(n^3 + m)$ элементарных арифметических операций над числами длиной $O(I)$, где I — длина входа (это показано в [18], см. также раздел 1.1.4 настоящей работы), получаем, что алгоритм $IsConsistentWithReduction$ является сильно полиномиальным.

5.3. Алгоритм решения систем

Дополним алгоритм проверки совместности системы $IsConsistentWithReduction$ так, чтобы в случае совместности системы дополненный алгоритм строил некоторое решение этой системы. Для этого при исключении каждой переменной y из системы сохраняются все ограничения $Ineqs_y$ этой системы, в которые эта переменная входит; и когда установлена совместность системы, подбирается значение для каждой основной переменной z, начиная с последней исключенной переменной, так, чтобы это значение удовлетворяло всем ограничениям $Ineqs_z$. Опишем это дополнение алгоритма $IsConsistentWithReduction$, указав шаги, которые нужно изменить или добавить.

Добавляется новый шаг (0).

(0) Создается изначально пустой стек $Stack$, элементами которого могут быть пары, состоящие из переменной и списка ограничений.

Действия алгоритма $IsConsistentWithReduction$ на шаге (2.1) заменяется на следующие.

(2.1) Если система S_{i-1} пуста, то всем основным переменным из V_{i-1} сопоставляется значение 0.

Добавляется новый шаг (2.6).

(2.6) Все ограничения системы S_{i-1}, содержащие только что исключенную переменную y, помещаются в стек вместе с этой переменной.

На шагах (3) и (4) выполняются следующие действия.

(3) Если система S_n пуста или выполнена при замене **0** на 0 и **1** на 1, то

(3.1) пока $Stack$ не пуст, из него выталкивается переменная z и список $Ineqs_z$ ограничений (в них входит z и другие переменные, но только такие, которым уже сопоставлены значения);

(3.3.1) ограничения $Ineqs_z$ разрешаются относительно z: получаются ограничения вида $c'_j \prec z$, $z \prec c''_k$, где c'_j, c''_k — рациональные числа[2], причем ограничения как одного, так и другого вида могут отсутствовать;

(3.3.2) выбираются (если существуют ограничения соответствующих видов) максимум Max из c'_j и минимум Min из c''_k;

(3.3.2) переменной z сопоставляется среднее арифметическое Max и Min, если они оба определены;
$Max + 1$, если определен только Max;
$Min - 1$, если определен только Min;
0, если ни Max, ни Min не определен;

(3.2) выдается ответ в виде списка пар из основных переменных и их значений.

(4) Иначе выдается ответ «несовместна».

Назовем получившийся алгоритм $FindSolutionWithReduction$.

Замечание 5.3.0.3. Аналогичным образом можно дополнить алгоритм $IsConsistent$ и получить алгоритм решения систем, основанный на обычном методе исключений переменных.

Очевидно (см. также раздел 12.2 в [46]), имеет место

Теорема 5.3.0.4. *Пусть дана непустая система S. Тогда алгоритм $FindSolutionWithReduction(S)$ выдает ответ «несовместна», ес-*

[2]Рациональное число представляется парой целых чисел.

ли S несовместна, и выдает некоторое решение системы S, если S совместна.

В дополнение к действиям алгоритма $IsConsistentWithReduction$ алгоритм $FindSolutionWithReduction$ в худшем случае решает $O(n)$ ограничений n раз, причем операции производятся над рациональными числами длиной $O(nL)$. Этот дополнительный вклад оценивается как $O(n^4L^2)$, и алгоритм $FindSolutionWithReduction$ имеет временную сложность $O(m \log n + mL^2 + n^3 \log n + n^4L^2)$, где m — число неравенств, n — число переменных, L — сумма длин всех коэффициентов исходной системы (см. также раздел 5.2.3). Таким образом, как и для алгоритма проверки совместности, заключаем, что алгоритм решения систем сильно полиномиален.

5.4. Программная реализация алгоритма решения систем

Как алгоритм $IsConsistentWithReduction$ проверки совместности систем, так и алгоритм $FindSolutionWithReduction$ решения систем реализованы на языке программирования Java в виде интерфейса прикладного программирования (ИПП). Опишем эту реализацию с точки зрения объектной декомпозиции, поскольку шаги этих и вспомогательных алгоритмов подробно описаны в предыдущих разделах.

5.4.1. Общая структура

ИПП для решения систем помещен в пакет `linineq2` со следующими основными классами.

`Variable`	Представляет переменную с именем-строкой `java.lang.String`.

`RationalNumber`	Представляет рациональное число парой длинных целых чисел `java.math.BigInteger` (используется только при нахождении решения системы).
`Inequality`	Представляет двучленное линейное неравенство (ограничение) с длинными целыми коэффициентами `java.math.BigInteger`.
`InequalitySystem`	Представляет систему неравенств как список объектов `Inequality`.
`EliminationAlgorithm`	Реализует обычный метод исключений переменных, причем позволяет подклассам менять представление и способ построения системы при исключении переменной.
`EliminationAlgorithm WithReduction`	Реализует алгоритмы $IsConsistentWithReduction$ и $FindSolutionWithReduction$.
`EliminationAlgorithm. SystemRep`	Абстрактный класс, задающий представление системы, которое используется в алгоритмах, основанных на методе исключений переменных.
`EliminationAlgorithm. ListSystemRep`	Представляет систему в виде списка объектов `Inequality`.

`EliminationAlgorithm WithReduction. MapSystemRep`	Представляет систему в виде отображения, сопоставляющего паре переменных множество объектов `Inequality`, содержащих эту пару переменных. Такое отображение основано на сбалансированном дереве поиска с ключами, являющимися парами переменных (см. раздел 5.2.2).

5.4.2. Декомпозиция алгоритмов

В разработанном ИПП для решения систем реализованы алгоритмы $IsConsistentWithReduction$ и $FindSolutionWithReduction$. В этих алгоритмах задействован вспомогательный алгоритм $IsConsistent$. Итак, имеются 3 алгоритма, основанных на методе исключений переменных. Эти алгоритмы различны, но в них есть похожие, иногда одинаковые шаги. Задача, которую мы ставили перед собой при объектно-ориентированном проектировании и реализации этих алгоритмов, заключается в том, чтобы не дублировать программный код, а вычленить общее поведение этих алгоритмов.

В классе `EliminationAlgorithmWithReduction`, реализованы алгоритмы $IsConsistentWithReduction$ и $FindSolutionWithReduction$. Этот класс является наследником класса `EliminationAlgorithm`. В последнем реализованы алгоритм $IsConsistent$ и алгоритм решения систем с помощью обычного метода исключений переменных. (См. диаграмму классов на рис. 5.1.)

Для проверки совместности системы используется метод

```
boolean isConsistent(),
```

определенный в классе `EliminationAlgorithm`.

Для решения системы используется метод

```
java.util.Map<Variable,RationalNumber> findSolution(),
```

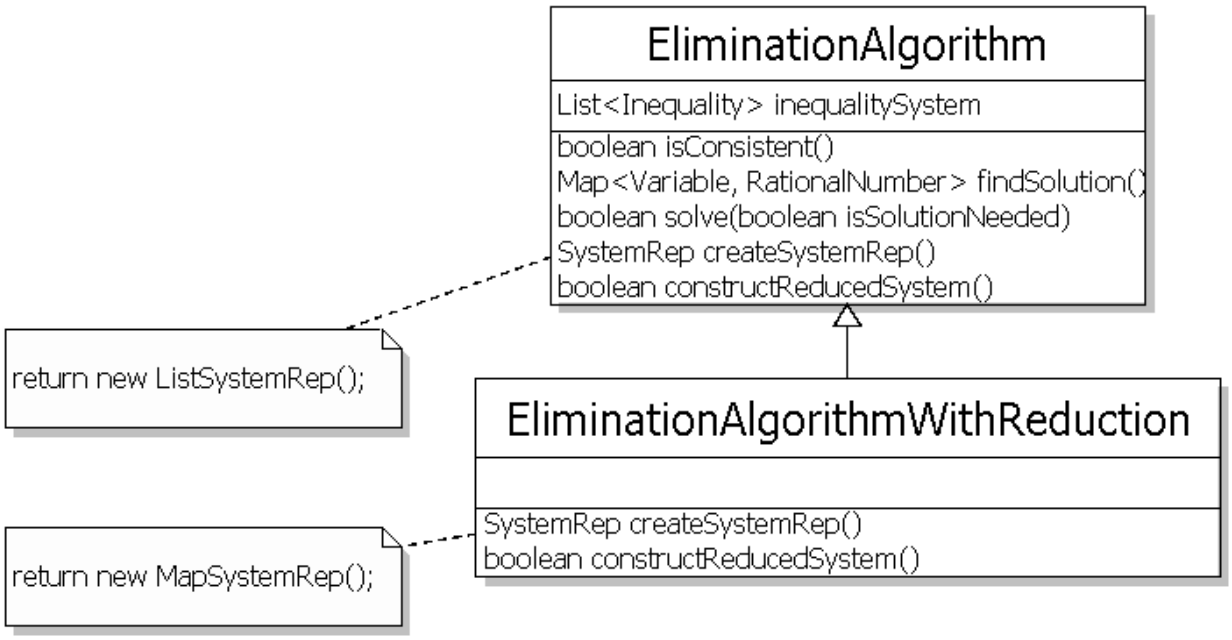


Рис. 5.1. Диаграмма классов, изображающая алгоритмы решения системы.

также определенный в классе `EliminationAlgorithm`.

Оба эти метода поручают всю работу методу

```
boolean solve(boolean isSolutionNeeded),
```

который решает систему или только проверяет систему на совместность в зависимости от значения параметра `isSolutionNeeded`. Если `isSolutionNeeded` — истина, то исключаемая переменная с ограничениями, в которые она входит, помещается в специально созданный для этого стек, а когда установлена совместность системы, подбирается значение для каждой основной переменной (так, как описано в разделе 5.3). Тем самым не происходит дублирование кода, которое могло бы быть вызвано написанием разного, но похожего кода для алгоритмов решения и проверки совместности.

Теперь объясним, как вычленяется общее поведение алгоритмов, основанных на обычном методе исключений переменных, и алгоритмов, основанных на том же методе, но с удалением лишних неравенств. Ниже приведен код метода `EliminationAlgorithm.solve()`.

```
protected boolean solve(boolean isSolutionNeeded) {
    assert (isSolutionNeeded ? solution != null : true) :
            "the field solution must not be null";

    SystemRep system = createSystemRep();
    Set<Variable> vars = new TreeSet<Variable>();
    Stack<EliminatedVarAndIneqs> elimVarAndIneqStack =
            (isSolutionNeeded ? new Stack<EliminatedVarAndIneqs>() : null);

    for (Inequality ineq : inequalitySystem) {
        if (!system.add(ineq)) {
             // ineq is identically false, the system is inconsistent
             return false;
        }
        vars.add(ineq.getX());
        vars.add(ineq.getY());
    }
    vars.remove(Variable.ZERO);
    vars.remove(Variable.UNIT);
    // Thus initially vars is a set of all principal variables
    // occurring in the system. Then variables will be eliminated
    // from the current system and from this set.

    while (!vars.isEmpty()) { // elimination loop

        if (system.isEmpty()) {
            if (isSolutionNeeded) {
                for (Variable var : vars) {
                    // may put any number
                    solution.put(var, RationalNumber.ZERO);
                }
            }
            break;
        }

        assert !system.isEmpty();

        // Choose a Variable to eliminate from the system.
```

```
    // (May invoke next() since !vars.isEmpty().)
    Variable elimVar = vars.iterator().next();

    assert elimVar != null && !elimVar.isAuxiliary();

    // Construct the reduced system without elimVar.

    vars.remove(elimVar);
    SystemRep reducedSystem = createSystemRep();

    // inequalities with the variable to eliminate
    // that has a negative factor
    List<Inequality> ineqsWithNegElimVar = new LinkedList<Inequality>();
    // inequalities with the variable to eliminate
    // that has a positive factor
    List<Inequality> ineqsWithPosElimVar = new LinkedList<Inequality>();

    if (!constructReducedSystem(system, reducedSystem, elimVar,
            ineqsWithNegElimVar, ineqsWithPosElimVar)) {
        // an idendically false composition is constructed,
        // the system is inconsistent
        return false;
    }

    // reducedSystem has been constructed, solve it

    if (isSolutionNeeded) {
        elimVarAndIneqStack.push(new EliminatedVarAndIneqs(elimVar,
                ineqsWithNegElimVar, ineqsWithPosElimVar));
    }

    system = reducedSystem;
} // end of while(!vars.isEmpty())

boolean isConsistent = isConstTrue(system);

if (!isConsistent || !isSolutionNeeded) {
    return isConsistent;
```

```
    }

    assert isSolutionNeeded;

    // Construct a solution of the system.
    while (!elimVarAndIneqStack.isEmpty()) {
        EliminatedVarAndIneqs elimVarAndIneqs = elimVarAndIneqStack.pop();
        RationalNumber elimVarValue = findValue(elimVarAndIneqs.elimVar,
                elimVarAndIneqs.ineqsWithNegElimVar,
                elimVarAndIneqs.ineqsWithPosElimVar,
                solution);
        solution.put(elimVarAndIneqs.elimVar, elimVarValue);
    }

    assert isSolution(inequalitySystem, solution) :
            "constructed map is not a solution of the initial system: " +
            solution;
    assert isConsistent;

    return true;
} // end of solve()
```

Основные шаги вышеприведенного метода таковы. С помощью вызова `createSystemRep()` создается объект класса `SystemRep`, представляющий систему. Создается множество (объект класса `java.util.TreeSet`) для хранения переменных, причем итератор такого множества выдает переменные в соответствии с порядком, заданным в классе `Variable` путем реализации интерфейса `java.lang.Comparable` (см. [2, 65]). И также создается стек для хранения объектов класса `EliminatedVarAndIneqs`, состоящих из переменной и двух списков ограничений: первый — для ограничений с отрицательными коэффициентами при этой переменной, второй — с положительными. Затем объект, представляющий систему, заполняется ограничениями.

Далее начинается цикл, на каждом шаге которого производится исключение одной переменной. При этом вызов `createSystemRep()` создает объект класса `SystemRep`, представляющий очередную систему без исключа-

емой на данном шаге переменной. А само построение этой новой системы поручается методу `constructReducedSystem()`. Если требуется решение, то соответствующий элемент помещается в стек.

После завершения этого цикла проверяется, выполнена ли полученная система, содержащая только дополнительные переменные. Если требуется решение системы, то оно строится, начиная с последней исключенной переменной.

Таким образом, метод `EliminationAlgorithm.solve()` включает в себя действия, общие для алгоритмов, основанных на обычном методе исключений переменных, и алгоритмов, основанных на том же методе, но с удалением лишних неравенств.

Метод `EliminationAlgorithm.solve()` является *шаблонным методом* (см. [42]). Вызываемые из него методы `createSystemRep()` и `constructReducedSystem()` переопределены в классе `EliminationAlgorithmWithReduction`.

В классе `EliminationAlgorithm` метод `createSystemRep()` создает объект класса `ListSystemRep`, представляющий систему в виде списка ограничений. (Вспомним, что алгоритм $IsConsistent$ применяется только для систем с не более чем двумя основными переменными. Поэтому не стоит использовать более сложную структуру данных.)

В классе `EliminationAlgorithmWithReduction` метод `createSystemRep()` создает объект класса `MapSystemRep`, представляющий систему в виде отображения. Такое отображение сопоставляет паре переменных множество ограничений, содержащих эту пару переменных, и основано на сбалансированном дереве поиска, в котором ключами являются пары переменных (см. также раздел 5.2.2).

Иерархия классов, представляющих системы, изображена на рис. 5.2. (См. также рис. 5.1 на с. 148: на этом рисунке наглядно показаны реализации метода `createSystemRep()`).

Для добавления ограничения в объект класса `SystemRep` служит метод

```
boolean add(Inequality ineq).
```

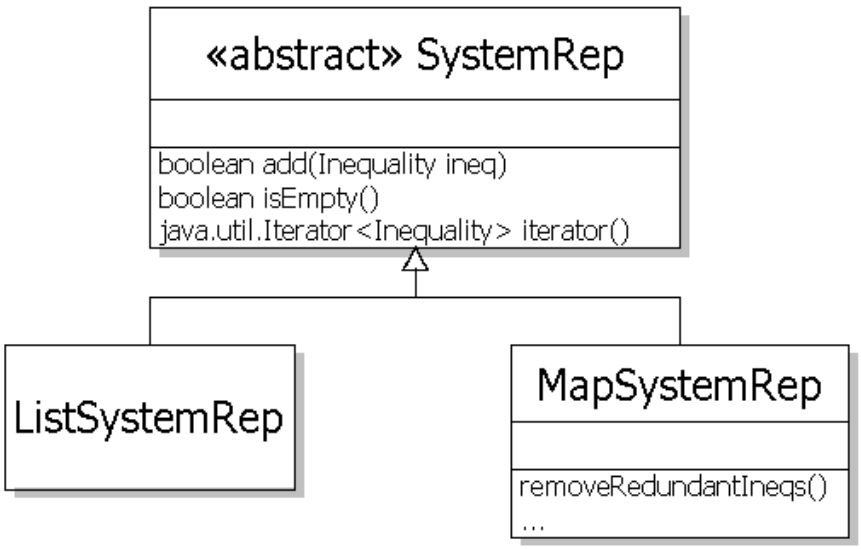


Рис. 5.2. Диаграмма классов, изображающая представления системы.

Метод `MapSystemRep.add()` при добавлении ограничения производит удаление лишних ограничений, короче говоря, этот метод представляет собой реализацию алгоритма *AddConstraint*.

Метод `constructReducedSystem()`, определенный в классе `EliminationAlgorithm`, производит шаги (2.3)–(2.4) алгоритма *IsConsistent*, оперируя с объектом класса `ListSystemRep`. Метод `constructReducedSystem()`, переопределенный в классе `EliminationAlgorithmWithReduction`, производит шаги (2.3)–(2.5) алгоритма *IsConsistentWithReduction*, оперируя с объектом класса `MapSystemRep`.

Таким образом, общее поведение нескольких реализованных алгоритмов вычленено и представлено кодом, который является общим каркасом для всех этих алгоритмов.

5.4.3. Предоставляемый программный интерфейс

Использование ИПП для решения систем заключается в создании системы (объекта класса `InequalitySystem`) и вызове метода, решающего систему или только проверяющего ее совместность.

Для создания объекта, представляющего переменную, служит следующий статический метод класса `Variable`:

```
Variable valueOf(java.lang.String name),
```

параметром этого метода является имя переменной.

Объект, представляющий рациональное число, создается с помощь статических методов

```
RationalNumber getInstance(java.math.BigInteger numerator,
                           java.math.BigInteger denominator);
RationalNumber getInstance(java.math.BigInteger numerator)
```

класса `RationalNumber`. Первый метод принимает в качестве параметров числитель и знаменатель рационального числа, представленные длинными целыми числами `java.math.BigInteger`. Второй метод является сокращенной формой первого, где знаменатель равен 1.

Ограничение создается с помощью одного из следующих статических методов класса `Inequality`:

```
Inequality getInstance(java.math.BigInteger a, Variable x,
                       java.math.BigInteger b, Variable y,
                       Inequality.InequalitySign sign);
Inequality getInstance(java.math.BigInteger a, Variable x,
                       java.math.BigInteger b,
                       Inequality.InequalitySign sign);
Inequality getInstance(java.math.BigInteger a, Variable x,
                       Inequality.InequalitySign sign).
```

Первому методу в передаются коэффициент при первой переменной, эта переменная, коэффициент при второй переменной, вторая переменная и знак неравенства — одна из констант перечисления `Inequality.InequalitySign`: `LESS` ($<$) или `LESS_OR_EQUAL` ($\leq$). Второй и третий методы являются очевидными сокращенными формами первого.

Также ограничение можно создать, проведя синтаксический анализ строки с помощью статического метода класса `Inequality`:

```
Inequality getInstance(java.lang.String s).
```

Правильно построенные ограничения задаются следующей формой Бэкуса-Наура.

```
Ограничение = ПервоеСлагаемое Слагаемое ЗнакНеравенства "0"
Ограничение = ПервоеСлагаемое ЗнакНеравенства "0"
FirstTerm = СлагаемоеБезЗнака | "-" СлагаемоеБезЗнака
Слагаемое = "+" СлагаемоеБезЗнака | "-" СлагаемоеБезЗнака
СлагаемоеБезЗнака = НеотрицательноеЦелое "*" Переменная |
    НеотрицательноеЦелое | Переменная |
    НеотрицательноеЦелое "*" ДополнительнаяПеременная
ДополнительнаяПеременная = "0" | "1"
Переменная = Буква БуквыИЦифры
ЗнакНеравенства = "<" | "<="
```

Нетерминалы `НеотрицательноеЦелое`, `БуквыИЦифры` и `Буква` имеют очевидный смысл, если уточнить, что используются десятичные цифры и буквы английского алфавита. Правила для этих нетерминалов не приводятся в целях сокращения записи.

Наконец, система создается с помощью конструктора класса `InequalitySystem`

```
InequalitySystem(java.util.List<Inequality> inequalities),
```

которому передается список объектов класса `Inequality`. Для проверки совместности системы служит метод

```
boolean isConsistent(),
```

который возвращает `true` тогда и только тогда, когда система совместна. Для решения системы используется метод

```
java.util.Map<Variable, RationalNumber> findSolution(),
```

который возвращает возвращает `null`, если система несовместна, иначе возвращает решение системы — отображение, сопоставляющее переменной ее значение.

5.4.4. Экспериментальные результаты

Проведены эксперименты по сравнению производительности

- реализованного в настоящей работе алгоритма, представленного методом `InequalitySystem.findSolution()`, и
- алгоритма, представленного функцией `FindInstance` системы компьютерной алгебры Mathematica 5.2 for Students [71].

На вход этим алгоритмам подавались случайным образом сгенерированные системы линейных двучленных неравенств с целыми 64-битовыми коэффициентами. Программа, осуществляющая случайную генерацию систем и вызов того и другого алгоритма, приведена в приложении Б.

Эксперименты проводились на компьютере с процессором Pentium M с тактовой частотой 1,6 ГГц и оперативной памятью объемом 512 Мб под управлением операционной системы Windows XP. Результаты экспериментов приведены в таблице 5.1. Эти результаты говорят о том, что для решения систем линейных двучленных неравенств реализованный в данной работе алгоритм намного эффективнее алгоритма, используемого в системе компьютерной алгебры Mathematica, и позволяет решать системы неравенств бóльших размеров.

Число переменных	Число неравенств	Среднее время решения системы в `linineq2`, секунды	Среднее время решения системы в Mathematica, секунды
10	50	0,07	0,33
100	500	0,16	1,4
500	2500	0,4	123
1000	5000	0,9	1100
5000	25000	5	не заканчивает работу за 12 часов
10000	50000	10	не заканчивает работу за 12 часов
50000	250000	44	не заканчивает работу за 12 часов

Таблица 5.1. Результаты сравнения производительности алгоритмов решения систем в `linineq2` и `FindInstance` в Mathematica.

Глава 6.

Заключение

Основные результаты

В данной работе получены следующие основные результаты.

1. Сформулировано секвенциальное исчисление для бесконечнозначной предикатной логики Лукасевича, расширенной модификаторами типа «очень», которые восходят к нечеткой логике Заде.

2. Исследованы свойства предложенного секвенциального исчисления, служащие теоретической основой для разработки алгоритма поиска вывода в этом исчислении.

3. Разработан алгоритм поиска вывода в предложенном секвенциальном исчислении. Доказаны свойства разработанного алгоритма, отражающие различные аспекты его корректности.

4. Алгоритм поиска вывода реализован в виде интерфейса прикладного программирования.

5. Уточнен алгоритм решения систем линейных двучленных неравенств, используемый для распознавания некоторых аксиом введенного секвенциального исчисления. Получена оценка временной сложности этого алгоритма в формальной вычислительной модели.

6. Алгоритм решения систем линейных двучленных неравенств реализован в виде интерфейса прикладного программирования.

Научная новизна результатов

Все основные результаты диссертации являются новыми. Следует отметить, что до данной работы для бесконечнозначной предикатной логики Лукасевича были известны лишь исчисления гильбертовского типа, и не были разработаны ни теоретические основы, ни программные средства для автоматического доказательства в этой логике.

Теоретическая и практическая ценность результатов

Предложенная логика может быть использована для представления нечетких знаний. Сформулированное секвенциальное исчисление может использоваться для доказательства не только во введенной логике, но и в бесконечнозначной предикатной логике Лукасевича, что значительно эффективнее, чем использование исчислений гильбертовского типа.

Реализованный интерфейс прикладного программирования (ИПП) для поиска вывода может применяться, например, в исследовательских целях для автоматического доказательства как в предложенной логике, так и в логике Лукасевича. Этот ИПП может служить ядром дедуктивной системы, основанной на любой из упомянутых логик.

Реализованный (также в виде ИПП) алгоритм решения систем линейных двучленных неравенств может использоваться для решения задач бóльших размеров, чем позволяют системы компьютерной алгебры, которые предоставляют алгоритмы решения более общих задач.

Литература

[1] Алгорифм машинного поиска естественного логического вывода в исчислении высказываний. / Н. А. Шанин, Г. В. Давыдов, С. Ю. Маслов и др. — М., Л.: Наука, 1965.

[2] *Арнолд .К, Гослинг Дж., Холмс Д.* Язык программирования Java. / Пер. с англ. — М.: Издательский дом «Вильямс», 2001.

[3] *Ахо А., Сети Р., Ульман Дж.* Компиляторы: принципы, технологии и инструменты. / Пер. с англ. — М.: Издательский дом «Вильямс», 2001.

[4] *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. / Пер. с англ. — М.: Мир, 1979.

[5] *Ахо А., Хопкрофт Дж., Ульман Дж.* Структуры данных и алгоритмы. / Пер. с англ. — М.: Издательский дом «Вильямс», 2000.

[6] *Бадд Т.* Объектно-ориентированное программирование в действии. / Пер. с англ. — СПб: Питер, 1997.

[7] *Биркгоф Г.* Теория решеток. / Пер. с англ. — М.: Наука, 1984.

[8] *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. / Пер. с англ. — Второе изд. — М.: Издательство Бином, 1998.

[9] *Буч Г., Рамбо Дж., Джекобсон А.* Язык UML. Руководство пользователя. / Пер. с англ. — М.: ДМК Пресс, 2004.

[10] *Вирт Н.* Алгоритмы и структуры данных. / Пер. с англ. — 2-е изд. — СПб: Невский Диалект, 2001.

[11] *Герасимов А. С.* Бесконечнозначная предикатная логика со связкой для усиления утверждений. // Современная логика: проблемы теории, истории и применения в науке: Материалы IX Общероссийской научной конференции (Санкт-Петербург, 2006). — СПб: 2006. — С. 348–350.

[12] *Герасимов А. С.* Предикатная логика на основе секвенциального исчисления, предназначенная для моделирования непрерывных шкал. // Десятая национальная конференция по искусственному интеллекту с международным участием КИИ-06 (Обнинск, 2006): Труды конференции. — М.: Физматлит, 2006. — С. 339–347.

[13] *Герасимов А. С.* Программная реализация поиска доказательств в бесконечнозначной предикатной логике, основанной на линейных неравенствах. // Материалы XVI Международной школы-семинара «Синтез и сложность управляющих систем» (Санкт-Петербург, 2006). / Под ред. О. Б. Лупанова. — М.: Изд-во механико-математического факультета МГУ, 2006. — С. 30–35.

[14] *Герасимов А. С.* Разработка ИПП для решения задачи определения совместности систем линейных двучленных неравенств. // Технологии Microsoft в теории и практике программирования: Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада. — СПб: Изд-во Политехн. ун-та, 2005. — С. 161–162.

[15] *Герасимов А. С., Косовский Н. К.* Истинно полиномиальный алгоритм определения совместности систем линейных двучленных неравенств. // Устойчивость и процессы управления. Труды международной конференции (Санкт-Петербург, 2005). / Под ред. Д. А. Овсянникова, Л. А. Петросяна. — СПб: СПбГУ, НИИ ВМ и ПУ, ООО ВММ, 2005. — С. 779–785.

[16] *Герасимов А. С., Косовский Н. К.* Оценка сложности истинно полиномиального алгоритма проверки совместности систем линейных двучленных неравенств. // *Вестник С.-Петербург. ун-та. Сер. 10.* — 2006. — № 2. — С. 16–21.

[17] *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. / Пер. с англ. — М.: Мир, 1982.

[18] *Давыдок Д. В.* О совместности систем двучленных линейных неравенств. // Косовский Н. К., Тишков А. В. Логики конечнозначных предикатов на основе неравенств. — СПб: Издательство С.-Петерб. университета, 2000. — С. 246–268.

[19] *Заде Л.* Понятие лингвистической переменной и его применение к принятию приближенных решений. / Пер. с англ. — М.: Мир, 1976.

[20] *Зиглер К.* Методы проектирования программных систем. / Пер. с англ. — М.: Мир, 1985.

[21] *Кангер С.* Упрощенный метод доказательства для элементарной логики. // Математическая теория логического вывода. / Под ред. А. В. Идельсона, Г. Е. Минца. — М.: Наука, 1967. — С. 200–207.

[22] *Клини С. К.* Введение в метаматематику. / Пер. с англ. — М.: Издательство иностранной литературы, 1957.

[23] *Клини С. К.* Математическая логика. / Пер. с англ. — М.: Мир, 1973.

[24] *Колмогоров А. Н., Драгалин А. Г.* Математическая логика. — М.: Едиториал УРСС, 2004.

[25] *Комендантский В. Е.* Метод резолюций в смешанной логике Поста. // Труды семинара логического центра ИФ РАН, вып. XVI. — 2002. — С. 64–74.

[26] *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. / Пер. с англ. — М.: МЦНМО, 1999.

[27] *Косовский Н. К.* Уровневые логики. // Записки научных семинаров Петербургского отделения Математического института РАН. — Т. 220. — СПб: Наука, 1995. — С. 72–82.

[28] *Косовский Н. К., Тишков А. В.* Полиномиальные алгоритмы установления совместности в рациональных и целых числах систем строгих и нестрогих линейных неравенств. // Актуальные проблемы современной математики. — Т. 3. — Новосибирск: Изд-во НИИ МИОО Новосибирского гос. ун-та, 1997. — С. 95–100.

[29] *Косовский Н. К., Тишков А. В.* Логики конечнозначных предикатов на основе неравенств. — СПб: Издательство С.-Петерб. университета, 2000.

[30] *Лисков Б., Гатэг Дж.* Использование абстракций и спецификаций при разработке программ. / Пер. с англ. — М.: Мир, 1989.

[31] *Макконнелл С.* Совершенный код. / Пер. с англ. — М.: Издательско-торговый дом «Русская Редакция», 2005.

[32] *Маслов С. Ю.* Обратный метод установления выводимости для логических исчислений. // Труды Математического института АН СССР им. В. А. Стеклова. — Т. 98. — М.: Наука, 1968. — С. 26–87.

[33] *Маслов С. Ю.* Теория дедуктивных систем и ее применения. — М.: Радио и связь, 1986.

[34] *Маслов С. Ю., Минц Г. Е.* Теория поиска вывода и обратный метод. // Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. — М.: Наука, 1983. — С. 291–314.

[35] *Матулис В. А.* Два варианта классического исчисления предикатов без структурных правил вывода. // Доклады Академии наук СССР. — 1962. — Т. 147. — С. 1029–1031.

[36] *Мендельсон Э.* Введение в математическую логику. / Пер. с англ. — М.: Наука, 1984.

[37] *Новак В., Перфильева И., Мочкорж И.* Математические принципы нечеткой логики. / Пер. с англ. — М.: Физматлит, 2006.

[38] *Одинцов И. О.* Профессиональное программирование. Системный подход. — 2-е изд. — СПб: БХВ-Петербург, 2004.

[39] *Оревков В. П.* Обратный метод поиска вывода. // Адаменко А. Н., Кучуков А. М. Логическое программирование и Visual Prolog. — СПб: БХВ-Петербург, 2003. — С. 952–965.

[40] *Пападимитриу Х., Стайглиц К.* Комбинаторная оптимизация. Алгоритмы и сложность. / Пер. с англ. — М.: Мир, 1985.

[41] *Парис Дж., Харрингтон Л.* Математическая неполнота а арифметике Пеано. // Справочная книга по математической логике. / Под ред. Дж. Барвайса. Ч. IV. Теория доказательств и конструктивная математика. / Пер. с англ. — М.: Наука, 1983. — С. 319–327.

[42] Приемы объектно-ориентированного проектирования. Паттерны проектирования. / Пер. с англ. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. — СПб: Питер, 2001.

[43] *Роджерс Х.* Теория рекурсивных функций и эффективная вычислимость. / Пер. с англ. — М.: Мир, 1972.

[44] *Смирнова Е. С.* Разработка и реализация алгоритма обработки нечетких данных в многоуровневых логиках: Диссертация на соиск. учен. степ. канд. физ.-мат. наук. / Санкт-Петербургский государственный университет. — Санкт-Петербург, 2002.

[45] *Соммервилл И.* Инженерия программного обеспечения. / Пер. с англ. — 6-е изд. — М.: Издательский дом «Вильямс», 2002.

[46] *Схрейвер А.* Теория линейного и целочисленного программирования. / Пер. с англ. — М.: Мир, 1991.

[47] *Фаулер М., Скотт К.* UML в кратком изложении. Применение стандартного языка объектного моделирования. / Пер. с англ. — М.: Мир, 1999.

[48] *Хантер Р.* Проектирование и конструирование компиляторов. / Пер. с англ. — М.: Финансы и статистика, 1984.

[49] *Хачиян Л. Г.* Полиномиальные алгоритмы в линейном программировании. // *Журнал вычислительной математики и математической физики.* — 1980. — № 1. — С. 51–68.

[50] *Aguzzoli S., Ciabattoni A.* Finiteness in infinite-valued Łukasiewicz logic // *Journal of Logic, Language and Information.* — 2000. — Vol. 9, no. 1. — Pp. 5–29.

[51] *Baaz M., Fermüller C. G., Salzer G.* Automated deduction for many-valued logics. // Handbook of Automated Reasoning. / Ed. by A. Robinson, A. Voronkov. — Elsevier, 2001. — Vol. 2. — Pp. 1355–1402.

[52] *Chandru V., Rao M. R.* Linear programming. // Algorithms and theory of computation handbook. / Ed. by M. J. Atallah. — CRC Press, 1999.

[53] *Ciabattoni A., Fermüller C. G., Metcalfe G.* Uniform rules and dialogue games for fuzzy logics. // LPAR / Ed. by F. Baader, A. Voronkov. — Vol. 3452 of *Lecture Notes in Computer Science.* — Springer, 2004. — Pp. 496–510.

[54] *Cohen E., Megiddo N.* Improved algorithms for linear inequalities with two variables per inequality. // *SIAM Journal on Computing.* — 1994. — Vol. 23, no. 6. — Pp. 1313–1347.

[55] *Degtyarev A., Voronkov A.* Equality reasoning in sequent-based calculi. //

Handbook of Automated Reasoning. / Ed. by A. Robinson, A. Voronkov. — Elsevier, 2001. — Vol. 1. — Pp. 611–706.

[56] *Degtyarev A., Voronkov A.* Kanger's choices in automated reasoning. // Collected Papers of Stig Kanger with Essays on his Life and Work. / Ed. by G. Holmström-Hintikka, S. Linström, R. Slivinski. — Kluwer Academic Publishers, 2001. — Vol. II. — Pp. 53–68.

[57] *Fitting M.* First-Order Logic and Automated Theorem Proving. — Second edition. — Springer, 1996.

[58] *Gottwald S.* A Treatise on Many-Valued Logics. — Baldock: Research Studies Press, 2001.

[59] *Hähnle R.* Many-valued logic and mixed integer programming. // *Annals of Mathematics and Artificial Intelligence.* — 1994. — Vol. 12, no. 3-4. — Pp. 231–263.

[60] *Hájek P.* Metamathematics of Fuzzy Logic. — Dordrecht: Kluwer Academic Publishers, 1998.

[61] *Hájek P.* What is mathematical fuzzy logic. // *Fuzzy Sets and Systems.* — 2006. — Vol. 157, no. 5. — Pp. 597–603.

[62] *Hay L.* Axiomatization of the infinite-valued predicate calculus. // *Journal of Symbolic Logic.* — 1963. — Vol. 28, no. 1. — Pp. 77–86.

[63] *Hochbaum D. S., Naor J.* Simple and fast algorithms for linear and integer programs with two variables per inequality. // *SIAM Journal on Computing.* — 1994. — Vol. 23, no. 6. — Pp. 1179–1192.

[64] Java 2 Platform Standard Edition 5.0. `http://java.sun.com/j2se/1.5.0`.

[65] Java 2 Platform Standard Edition 5.0 API Specification. `http://java.sun.com/j2se/1.5.0/docs/api`.

[66] Javadoc, an API documentation generator.
`http://java.sun.com/j2se/javadoc`.

[67] J/Link, a toolkit that integrates Mathematica and Java.
`http://www.wolfram.com/solutions/mathlink/jlink`.

[68] *Karmarkar N.* A new polynomial-time algorithm for linear programming. // *Combinatorica.* — 1984. — Vol. 4, no. 4. — Pp. 373–396.

[69] *Łukasiewicz J.* O logice trójwartościowej. // *Ruch Filozoficny.* — 1920. — Vol. 5. — Pp. 170–171.

[70] Maple, a computer algebra system.
`http://www.maplesoft.com/products/maple`.

[71] Mathematica, a computer algebra system.
`http://www.wolfram.com/products/mathematica`.

[72] *Megiddo N.* Towards a genuinely polynomial algorithm for linear programming. // *SIAM Journal on Computing.* — 1983. — Vol. 12, no. 2. — Pp. 347–353.

[73] *Metcalfe G., Olivetti N., Gabbay D. M.* Łukasiewicz logic: from proof systems to logic programming. // *Logic Journal of the IGPL.* — 2005. — Vol. 13, no. 5. — Pp. 561–585.

[74] *Metcalfe G., Olivetti N., Gabbay D. M.* Sequent and hypersequent calculi for abelian and Łukasiewicz logics. // *ACM Transactions on Computational Logic.* — 2005. — Vol. 6, no. 3. — Pp. 578–613.

[75] *Mundici D., Olivetti N.* Resolution and model building in the infinite-valued calculus of Łukasiewicz. // *Theoretical Computer Science.* — 1998. — Vol. 200, no. 1-2. — Pp. 335–366.

[76] *Olivetti N.* Tableaux for Łukasiewicz infinite-valued logic. // *Studia Logica.* — 2003. — Vol. 73, no. 1. — Pp. 81–111.

[77] *Pavelka J.* On fuzzy logic I, II, III. // *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik.* — 1979. — Vol. 25. — Pp. 45–52, 119–134, 447–464.

[78] *Prijatelj A.* Bounded contraction and Gentzen-style formulation of Łukasiewicz logics. // *Studia Logica.* — 1996. — Vol. 57. — Pp. 431–456.

[79] *Scarpellini B.* Die Nichtaxiomatisierbarkeit des Unendlichwertigen Pradikatenkalkuls von Lukasiewicz. // *Journal of Symbolic Logic.* — 1962. — Vol. 27, no. 2. — Pp. 159–170.

[80] *Wagner H.* A new resolution calculus for the infinite-valued propositional logic of Łukasiewicz. // Int. Workshop on First-Order Theorem Proving. / Ed. by R. Caferra, G. Salzer. — 1998. — Pp. 234–243.

[81] *Zadeh L. A.* Fuzzy sets. // *Information and Control.* — 1965. — Vol. 8, no. 3. — Pp. 338–353.

[82] *Zadeh L. A.* Fuzzy logic and approximate reasoning. // *Synthese.* — 1975. — Vol. 30. — Pp. 407–428.

Приложение А.

Использование ИПП для автоматического поиска вывода

А.1. Программа, использующая разработанный ИПП для вывода

Ниже приводится исходный текст небольшой программы, использующей разработанный ИПП для поиска вывода. Приведенные в следующих разделах протоколы поиска вывода получены с помощью этой программы.

```
import prover.*;
import prover.calculus.*;
import prover.calculus.syntax.*;
import java.util.List;

public class Tester {

    private static class ProofPrinter implements ProofListener {
        // current number of an application of a rule
        private long ruleApplNumber;
```

```
public void proofSearchStarted(ProofEvent event) {
    System.out.println();
    System.out.println("Searching for a proof of the sequent:");
    System.out.println(event.getSkeleton().getRootSequent());
    System.out.println();
}

public void ruleApplied(ProofEvent event) {
    Tactics.RuleChoice rc = event.getRuleChoice();

    System.out.println(++ruleApplNumber +
            ". Inference rule " + rc.getRule() +
            " has been applied backwards to the sequent:");
    System.out.println(rc.getNode().getSequent());
    System.out.println("Main formula:");
    System.out.println(rc.getMainFormula());
    System.out.println("Subformula:");
    System.out.println(rc.getSubformula());
    System.out.println("Premises:");
    for (ProofSkeleton.Node node : rc.getNode().getChildren()) {
         System.out.println(node.getSequent());
         System.out.println();
    }
}

public void closingSkeleton(ProofEvent event) {
    ProofSkeleton skeleton = event.getSkeleton();

    System.out.println("Trying to close the proof skeleton " +
            "with the following leaf sequents:");
    for (ProofSkeleton.Node node : skeleton.getLeaves()) {
        System.out.println(node.getSequent());
        System.out.println();
    }

    List<Metavariable> mvs = skeleton.getMetavariables();
    if (mvs != null && !mvs.isEmpty()) {
        System.out.println("Metavariables and their substitution sets:");
```

```
        for (Metavariable mv : mvs) {
            System.out.println(mv.getName() + " in " +
                               mv.getSubstitutionSet());
        }
    }
}

public void metavariablesInstantiated(ProofEvent event) {
    List<Metavariable> mvs = event.getSkeleton().getMetavariables();
    if (mvs != null && !mvs.isEmpty()) {
        System.out.println();
        System.out.println("All metavariables have been instantiated:");
        for (Metavariable mv : mvs) {
            System.out.println(mv.getName() + " = " + mv.getValue());
        }
    }
}

public void systemOfInequalitiesConstructed(ProofEvent event) {
    System.out.println();
    System.out.println("The system of linear inequalities has been " +
                       "constructed:");
    System.out.println(event.getInequalitySystem());
    System.out.println("from the sequent:");
    System.out.println(event.getSequent());
}

public void sequentTestedForAxiom(ProofEvent event) {
    boolean isAxiom = event.isSequentAxiom();
    System.out.println("The above mentioned sequent is " +
                       (isAxiom ? "" : "NOT ") + "an AXIOM.");
}

public void proofSearchFinished(ProofEvent event) {
    System.out.println();
    System.out.println("The proof search has been finished with" +
                       " the result: " + event.getProofStatus());
    List<InstantiationContext> unifiers = event.getUnifiers();
```

```
            if (unifiers != null && !unifiers.isEmpty()) {
                System.out.println("Unifiers:");
                for (InstantiationContext unf : unifiers) {
                    System.out.println(unf);
                }
            }
        }
    } // end of class ProofPrinter

    public static void main(String[] args) {
        String s = ConsoleReader.readString();
        Parser parser = new Parser(s);
        Sequent sequent = null;

        try {
            sequent = parser.parse();
        } catch (Parser.ParseException exc) {
            exc.printStackTrace();
            System.err.println("Please correct the input sequent.");
            return;
        }

        if (args.length > 0) {
            Prover.setMathematicaKernelPathName(args[0]);
        }
        Prover prover = new Prover(sequent);
        prover.addProofListener(new ProofPrinter());
        prover.prove();
    }
}
```

A.2. Пример вывода

Прежде, чем привести пример автоматически построенного вывода, сделаем замечание, касающееся обозначений.

Замечание A.2.0.1. Обозначения правил вывода представляются в виде

строк таким образом: `Ip` соответствует $\prec_+$, `In` — $\prec_-$, `Qp` — q_p, `Qn` — q_n. Логические символы $\&$, $\vee$, $\forall$, $\exists$ в обозначениях правил представляются как `&`, `V`, `A`, `E` соответственно. Например, `IpQp&` соответствует $(\prec_+ q_p\&)$.

Обозначения правил вывода $(q_1 \cdot q_2)$, $(q_p \prec)$, $(q_n \prec)$, $(q_1 \cdot q_2 \prec)$, $(\prec q_1 \cdot q_2)$ представляются как `Q1*Q2`, `QpI`, `QnI`, `Q1*Q2I`, `IQ1*Q2` соответственно.

Формулы и секвенции представляются в виде строк так, как указано в разделе 4.3. Кроме того, формулы и секвенции, не помещающиеся в одной строке текста, продолжаются на следующих строках.

Приведем протокол поиска вывода формулы логики Lq: $(2/3 \cdot \forall x \exists y(P1[0,1](x)\,\&\,P2[0,1](x,y)) \prec (\forall x P1[0,1](x)\,\&\,\forall x \exists y P2[0,1](x,y)))$.

```
Searching for a proof of the sequent:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) <
(A x P1[0, 1](x) & A x E y P2[0, 1](x, y)))

1. Inference rule IpQp& has been applied backwards to the sequent:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) <
(A x P1[0, 1](x) & A x E y P2[0, 1](x, y)))
Main formula:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) <
(A x P1[0, 1](x) & A x E y P2[0, 1](x, y)))
Subformula:
(A x P1[0, 1](x) & A x E y P2[0, 1](x, y))
Premises:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < A x P1[0, 1](x))
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < A x E y P2[0, 1](x, y))

2. Inference rule IpQpA has been applied backwards to the sequent:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < A x P1[0, 1](x))
Main formula:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < A x P1[0, 1](x))
Subformula:
A x P1[0, 1](x)
Premises:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
```

```
3. Inference rule IpQpA has been applied backwards to the sequent:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < A x E y P2[0, 1](x, y))
Main formula:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < A x E y P2[0, 1](x, y))
Subformula:
A x E y P2[0, 1](x, y)
Premises:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))

4. Inference rule QpAIp has been applied backwards to the sequent:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
Main formula:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
Subformula:
A x E y (P1[0, 1](x) & P2[0, 1](x, y))
Premises:
(2/3*E y (P1[0, 1](?x_1) & P2[0, 1](?x_1, y)) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))

5. Inference rule QpEIp has been applied backwards to the sequent:
(2/3*E y (P1[0, 1](?x_1) & P2[0, 1](?x_1, y)) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
Main formula:
(2/3*E y (P1[0, 1](?x_1) & P2[0, 1](?x_1, y)) < P1[0, 1](!1))
Subformula:
E y (P1[0, 1](?x_1) & P2[0, 1](?x_1, y))
Premises:
(2/3*(P1[0, 1](?x_1) & P2[0, 1](?x_1, !3)) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))

6. Inference rule Qp&Ip has been applied backwards to the sequent:
(2/3*(P1[0, 1](?x_1) & P2[0, 1](?x_1, !3)) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
Main formula:
(2/3*(P1[0, 1](?x_1) & P2[0, 1](?x_1, !3)) < P1[0, 1](!1))
Subformula:
(P1[0, 1](?x_1) & P2[0, 1](?x_1, !3))
Premises:
```

```
(2/3*P1[0, 1](?x_1) < P1[0, 1](!1)), (2/3*P2[0, 1](?x_1, !3) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))

7. Inference rule QpAIp has been applied backwards to the sequent:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
Main formula:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
Subformula:
A x E y (P1[0, 1](x) & P2[0, 1](x, y))
Premises:
(2/3*E y (P1[0, 1](?x_2) & P2[0, 1](?x_2, y)) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))

8. Inference rule QpEIp has been applied backwards to the sequent:
(2/3*E y (P1[0, 1](?x_2) & P2[0, 1](?x_2, y)) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
Main formula:
(2/3*E y (P1[0, 1](?x_2) & P2[0, 1](?x_2, y)) < E y P2[0, 1](!2, y))
Subformula:
E y (P1[0, 1](?x_2) & P2[0, 1](?x_2, y))
Premises:
(2/3*(P1[0, 1](?x_2) & P2[0, 1](?x_2, !4)) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))

9. Inference rule Qp&Ip has been applied backwards to the sequent:
(2/3*(P1[0, 1](?x_2) & P2[0, 1](?x_2, !4)) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
Main formula:
(2/3*(P1[0, 1](?x_2) & P2[0, 1](?x_2, !4)) < E y P2[0, 1](!2, y))
Subformula:
(P1[0, 1](?x_2) & P2[0, 1](?x_2, !4))
Premises:
(2/3*P1[0, 1](?x_2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](?x_2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))

10. Inference rule IpQpE has been applied backwards to the sequent:
(2/3*P1[0, 1](?x_2) < E y P2[0, 1](!2, y)),
```

```
(2/3*P2[0, 1](?x_2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
Main formula:
(2/3*P1[0, 1](?x_2) < E y P2[0, 1](!2, y))
Subformula:
E y P2[0, 1](!2, y)
Premises:
(2/3*P1[0, 1](?x_2) < P2[0, 1](!2, ?y_3)),
(2/3*P1[0, 1](?x_2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](?x_2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))

11. Inference rule IpQpE has been applied backwards to the sequent:
(2/3*P1[0, 1](?x_2) < P2[0, 1](!2, ?y_3)),
(2/3*P1[0, 1](?x_2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](?x_2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
Main formula:
(2/3*P2[0, 1](?x_2, !4) < E y P2[0, 1](!2, y))
Subformula:
E y P2[0, 1](!2, y)
Premises:
(2/3*P1[0, 1](?x_2) < P2[0, 1](!2, ?y_3)),
(2/3*P1[0, 1](?x_2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](?x_2, !4) < P2[0, 1](!2, ?y_4)),
(2/3*P2[0, 1](?x_2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))

12. Inference rule IpQpE has been applied backwards to the sequent:
(2/3*P1[0, 1](?x_2) < P2[0, 1](!2, ?y_3)),
(2/3*P1[0, 1](?x_2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](?x_2, !4) < P2[0, 1](!2, ?y_4)),
(2/3*P2[0, 1](?x_2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
Main formula:
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
Subformula:
E y P2[0, 1](!2, y)
```

```
Premises:
(2/3*P1[0, 1](?x_2) < P2[0, 1](!2, ?y_3)),
(2/3*P1[0, 1](?x_2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](?x_2, !4) < P2[0, 1](!2, ?y_4)),
(2/3*P2[0, 1](?x_2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, ?y_5)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))

Trying to close the proof skeleton with the following leaf sequents:
(2/3*P1[0, 1](?x_1) < P1[0, 1](!1)), (2/3*P2[0, 1](?x_1, !3) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))

(2/3*P1[0, 1](?x_2) < P2[0, 1](!2, ?y_3)),
(2/3*P1[0, 1](?x_2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](?x_2, !4) < P2[0, 1](!2, ?y_4)),
(2/3*P2[0, 1](?x_2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, ?y_5)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))

Metavariables and their substitution sets:
?x_1 in [!1]
?x_2 in [!2]
?y_3 in [!4, !2]
?y_4 in [!4, !2]
?y_5 in [!4, !2]

All metavariables have been instantiated:
?x_1 = !1
?x_2 = !2
?y_3 = !4
?y_4 = !4
?y_5 = !4

The system of linear inequalities has been constructed:
1/3*a<0
1*a-2/3*b<0
-1*a<=0
1*a-1<=0
```

```
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!1) < P1[0, 1](!1)), (2/3*P2[0, 1](!1, !3) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
The above mentioned sequent is an AXIOM.

The system of linear inequalities has been constructed:
-2/3*a+1*b<0
1/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!2) < P2[0, 1](!2, !4)), (2/3*P1[0, 1](!2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](!2, !4) < P2[0, 1](!2, !4)),
(2/3*P2[0, 1](!2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, !4)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
The above mentioned sequent is an AXIOM.

All metavariables have been instantiated:
?x_1 = !1
?x_2 = !2
?y_3 = !4
?y_4 = !4
?y_5 = !2

The system of linear inequalities has been constructed:
1/3*a<0
1*a-2/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!1) < P1[0, 1](!1)), (2/3*P2[0, 1](!1, !3) < P1[0, 1](!1)),
```

```
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
The above mentioned sequent is an AXIOM.

The system of linear inequalities has been constructed:
-2/3*a+1*b<0
1/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!2) < P2[0, 1](!2, !4)), (2/3*P1[0, 1](!2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](!2, !4) < P2[0, 1](!2, !4)),
(2/3*P2[0, 1](!2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, !2)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
The above mentioned sequent is an AXIOM.

All metavariables have been instantiated:
?x_1 = !1
?x_2 = !2
?y_3 = !4
?y_4 = !2
?y_5 = !4

The system of linear inequalities has been constructed:
1/3*a<0
1*a-2/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!1) < P1[0, 1](!1)), (2/3*P2[0, 1](!1, !3) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
The above mentioned sequent is an AXIOM.

The system of linear inequalities has been constructed:
```

```
-2/3*a+1*b<0
1*c-2/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
-1*c<=0
1*c-1<=0
from the sequent:
(2/3*P1[0, 1](!2) < P2[0, 1](!2, !4)), (2/3*P1[0, 1](!2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](!2, !4) < P2[0, 1](!2, !2)),
(2/3*P2[0, 1](!2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, !4)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
The above mentioned sequent is NOT an AXIOM.

All metavariables have been instantiated:
?x_1 = !1
?x_2 = !2
?y_3 = !4
?y_4 = !2
?y_5 = !2

The system of linear inequalities has been constructed:
1/3*a<0
1*a-2/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!1) < P1[0, 1](!1)), (2/3*P2[0, 1](!1, !3) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
The above mentioned sequent is an AXIOM.

The system of linear inequalities has been constructed:
-2/3*a+1*b<0
1*c-2/3*b<0
```

```
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
-1*c<=0
1*c-1<=0
from the sequent:
(2/3*P1[0, 1](!2) < P2[0, 1](!2, !4)), (2/3*P1[0, 1](!2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](!2, !4) < P2[0, 1](!2, !2)),
(2/3*P2[0, 1](!2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, !2)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
The above mentioned sequent is NOT an AXIOM.

All metavariables have been instantiated:
?x_1 = !1
?x_2 = !2
?y_3 = !2
?y_4 = !4
?y_5 = !4

The system of linear inequalities has been constructed:
1/3*a<0
1*a-2/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!1) < P1[0, 1](!1)), (2/3*P2[0, 1](!1, !3) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
The above mentioned sequent is an AXIOM.

The system of linear inequalities has been constructed:
-2/3*a+1*b<0
1/3*c<0
-1*a<=0
1*a-1<=0
```

```
-1*c<=0
1*c-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!2) < P2[0, 1](!2, !2)), (2/3*P1[0, 1](!2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](!2, !4) < P2[0, 1](!2, !4)),
(2/3*P2[0, 1](!2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, !4)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
The above mentioned sequent is an AXIOM.

All metavariables have been instantiated:
?x_1 = !1
?x_2 = !2
?y_3 = !2
?y_4 = !4
?y_5 = !2

The system of linear inequalities has been constructed:
1/3*a<0
1*a-2/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!1) < P1[0, 1](!1)), (2/3*P2[0, 1](!1, !3) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
The above mentioned sequent is an AXIOM.

The system of linear inequalities has been constructed:
-2/3*a+1*b<0
1/3*c<0
-1*a<=0
1*a-1<=0
-1*c<=0
1*c-1<=0
```

```
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!2) < P2[0, 1](!2, !2)), (2/3*P1[0, 1](!2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](!2, !4) < P2[0, 1](!2, !4)),
(2/3*P2[0, 1](!2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, !2)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
The above mentioned sequent is an AXIOM.

All metavariables have been instantiated:
?x_1 = !1
?x_2 = !2
?y_3 = !2
?y_4 = !2
?y_5 = !4

The system of linear inequalities has been constructed:
1/3*a<0
1*a-2/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!1) < P1[0, 1](!1)), (2/3*P2[0, 1](!1, !3) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
The above mentioned sequent is an AXIOM.

The system of linear inequalities has been constructed:
-2/3*a+1*b<0
-2/3*c+1*b<0
-1*a<=0
1*a-1<=0
-1*c<=0
1*c-1<=0
-1*b<=0
1*b-1<=0
```

```
from the sequent:
(2/3*P1[0, 1](!2) < P2[0, 1](!2, !2)), (2/3*P1[0, 1](!2) < E y P2[0, 1](!2, y)),
(2/3*P2[0, 1](!2, !4) < P2[0, 1](!2, !2)),
(2/3*P2[0, 1](!2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, !4)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
The above mentioned sequent is NOT an AXIOM.

All metavariables have been instantiated:
?x_1 = !1
?x_2 = !2
?y_3 = !2
?y_4 = !2
?y_5 = !2

The system of linear inequalities has been constructed:
1/3*a<0
1*a-2/3*b<0
-1*a<=0
1*a-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!1) < P1[0, 1](!1)), (2/3*P2[0, 1](!1, !3) < P1[0, 1](!1)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P1[0, 1](!1))
The above mentioned sequent is an AXIOM.

The system of linear inequalities has been constructed:
-2/3*a+1*b<0
-2/3*c+1*b<0
-1*a<=0
1*a-1<=0
-1*c<=0
1*c-1<=0
-1*b<=0
1*b-1<=0
from the sequent:
(2/3*P1[0, 1](!2) < P2[0, 1](!2, !2)), (2/3*P1[0, 1](!2) < E y P2[0, 1](!2, y)),
```

```
(2/3*P2[0, 1](!2, !4) < P2[0, 1](!2, !2)),
(2/3*P2[0, 1](!2, !4) < E y P2[0, 1](!2, y)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < P2[0, 1](!2, !2)),
(2/3*A x E y (P1[0, 1](x) & P2[0, 1](x, y)) < E y P2[0, 1](!2, y))
The above mentioned sequent is NOT an AXIOM.

The proof search has been finished with the result: PROVABLE
Unifiers:
{?y_5=!4, ?x_2=!2, ?y_3=!4, ?y_4=!4, ?x_1=!1}
{?y_5=!2, ?x_2=!2, ?y_3=!4, ?y_4=!4, ?x_1=!1}
{?y_5=!4, ?x_2=!2, ?y_3=!2, ?y_4=!4, ?x_1=!1}
{?y_5=!2, ?x_2=!2, ?y_3=!2, ?y_4=!4, ?x_1=!1}
```

A.3. Пример представления нечетких знаний и вывода

Пусть имеются следующие нечеткие знания, выраженные предложениями естественного языка. (Это переработанный пример из [73], где приходится обходиться пропозициональными формулами при поиске доказательства; последнее было отмечено в конце раздела 1.1.3.)

A_1: Если человек обладает хорошими навыками общения, то он подходит для работы в отделе сбыта или консультационном отделе.
A_2: Довольно молодой человек не подходит для работы в отделе сбыта.
A_3: John молод.
A_4: John обладает отличными навыками общения.

Требуется ответить на вопрос: кто подходит для работы в консультационном отделе?

Введем предикаты:

- $Pr[0,1](x)$, выражающий, насколько x обладает навыками общения,
- $Pm[0,1](x)$, выражающий, насколько x подходит для работы в отделе сбыта,

- $Pc[0,1](x)$, выражающий, насколько x подходит для работы в консультационном отделе, и

- $Py[0,1](x)$, выражающий, насколько x молод.

Формализуем данные нечеткие знания, представив их в виде формул логики Lq.[1] (Отрезки истинностный значений опущены, John представлен предметной константой $\#J$.)

A_1: $\forall x(Pr(x) \prec (Pm(x) \vee Pc(x)))$
A_2: $\forall x((2/3 \cdot Py(x)\ \&\ Pm(x)) \prec 1/10)$
A_3: $(4/5 \prec Py(\#J))$
A_4: $(9/10 \prec Pr(\#J))$

Заданный вопрос сводится к поиску вывода формулы $\exists z((A_1 \& A_2 \& A_3 \& A_4) \prec Pc(z))$. (Для удобства чтения некоторые скобки в формуле опущены.)

При построении вывода (если вывод будет найден) будут также найдены унификаторы. Тогда объединение значений метапеременной $?z\_1$, соответствующей переменной z, из каждого унификатора будет ответом на вопрос.

Сокращенный протокол поиска вывода выглядит следующим образом.

```
Searching for a proof of the sequent:
E z ((((A x (Pr[0, 1](x) < (Pm[0, 1](x) V Pc[0, 1](x))) &
A x ((2/3*Py[0, 1](x) & Pm[0, 1](x)) < 1/10)) & (4/5 < Py[0, 1](#J))) &
(9/10 < Pr[0, 1](#J))) < Pc[0, 1](z))
...
8 контрприменений правил и проверка листовых секвенций заготовки вывода
на аксиоматичность не приводятся из-за довольно большой длины секвенций.
...
The proof search has been finished with the result: PROVABLE
Unifiers:
{?z_1=#J, ?x_2=#J, ?x_3=#J}
```

[1] Безусловно, можно предложить и другие формализации в зависимости от того, как трактовать нечеткие знания.

Итак, значение метапеременной $?z\_1$, соответствующей z, есть $\#J$ (т.е. John).

Приложение Б.

Использование ИПП для автоматического решения систем линейных двучленных неравенств

Ниже приводится исходный текст небольшой программы, использующей разработанный ИПП для решения систем линейных двучленных неравенств. Кроме того, что эта программа является примером использования разработанного ИПП, она служит для сравнения производительности алгоритма, представленного методом `InequalitySystem.findSolution()`, и алгоритма, представленного функцией `FindInstance` системы компьютерной алгебры Mathematica (см. раздел 5.4.4).

```
import linineq2.*;
import com.wolfram.jlink.*;
import java.util.Map;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
```

```
public class LinTester {

    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Usage: java LinTester " +
                    "pathToMathematicaKernel numberOfVariables " +
                    "numberOfInequalities");
        } else {
            mathematicaPath = args[0];
            int numVars = Integer.parseInt(args[1]);
            int numIneqs = Integer.parseInt(args[2]);
            solveRandomSystem(numVars, numIneqs);
        }
    }


    /*
     * Generates a random system of two-term linear inequalities
     * with 64-bit coefficients and solves the system using
     * linineq2.InequalitySystem.findSolution() and Mathematica's
     * FindInstance.
     */
    private static void solveRandomSystem(int numVars, int numIneqs) {
        String[] vars = new String[numVars];
        for (int i = 0; i < numVars; i++) {
             vars[i] = createNewVariable();
        }
        List<String> inequalities = new ArrayList<String>(numIneqs);
        Random random = new Random();

        for (int i = 0; i < numIneqs; i++) {
            String x = null;
            String y = null;
            long a = 0;
            long b = 0;

            a = random.nextLong();
```

```
            b = random.nextLong();
            x = vars[random.nextInt(numVars)];
            if (!(random.nextBoolean() && random.nextBoolean() &&
                    random.nextBoolean())) {
                // y a non-null variable with probalility (1 - 0.5^3)
                y = vars[random.nextInt(numVars)];
            }

            StringBuilder ineq = new StringBuilder();
            ineq.append(a);
            if (x != null) {
                ineq.append('*');
                ineq.append(x);
            }
            if (b >= 0) {
                ineq.append('+');
            }
            ineq.append(b);
            if (y != null) {
                ineq.append('*');
                ineq.append(y);
            }
            if (random.nextBoolean()) {
                ineq.append("<0");
            } else {
                ineq.append("<=0");
            }

            inequalities.add(ineq.toString());
        }

        solveSystemWithLinineq2(inequalities);
        solveSystemWithMathematica(inequalities, vars);
    }


    private static void solveSystemWithLinineq2(List<String> inequalities) {
        long startTime = System.currentTimeMillis();
```

```
    System.out.println("linineq2 start time = " + startTime);

    List<Inequality> system = new ArrayList<Inequality>(inequalities.size());
    for (String ineq : inequalities) {
        system.add(Inequality.getInstance(ineq));
    }

    Map<Variable, RationalNumber> solution =
            new InequalitySystem(system).findSolution();
    System.out.println("linineq2: consistent = " + (solution != null));

    long endTime = System.currentTimeMillis();
    long linineq2Time = endTime - startTime;
    System.out.println("linineq2 time = " + linineq2Time);
}


private static void solveSystemWithMathematica(List<String> inequalities,
                                               String[] vars) {
    long startTime = System.currentTimeMillis();
    System.out.println("Mathematica start time = " + startTime);

    openMathematicaLink();
    boolean isConsistent = !isInconsistentInMathematica(
            toMathematicaString(inequalities, vars));
    System.out.println("Mathematica: consistent = " + isConsistent);

    long endTime = System.currentTimeMillis();
    long mathematicaTime = endTime - startTime;
    System.out.println("Mathematica time = " + mathematicaTime);
    closeMathematicaLink();
}


/*
 * The number of variables that have been created by
 * the method createNewVariable().
 */
```

```
private static int numberOfCreatedVars;

/*
 * Creates a new variable.
 * Successive invocations of this method return variables:
 * a, b, ..., z, a1, b1, ..., z1, a2, b2, ..., z2, ...
 */
private static String createNewVariable() {
    String name = null;

    if (numberOfCreatedVars < 'z'-'a'+1) {
        name = String.valueOf((char)('a'+numberOfCreatedVars));
    } else {
        name = String.valueOf((char)('a'+numberOfCreatedVars%('z'-'a'+1))) +
                numberOfCreatedVars/('z'-'a'+1);
    }

    numberOfCreatedVars++;
    return name;
}


private static String mathematicaPath;
private static KernelLink mathematicaLink;

public static void openMathematicaLink() {
    if (mathematicaLink != null) {
        return;
    }

    String[] mlArgs = {"-linkmode", "launch", "-linkname",
                       mathematicaPath};
    try {
        mathematicaLink = MathLinkFactory.createKernelLink(mlArgs);
        mathematicaLink.discardAnswer();
    } catch (MathLinkException mle) {
        throw new Error("Fatal error opening Mathematica kernel link",
                        mle);
```

```
    }
}

public static void closeMathematicaLink() {
    mathematicaLink.close();
    mathematicaLink = null;
}

/*
 * Constructs a string that represents a call of Mathematica's FindInstance.
 * The format is like this: "FindInstance[3*x-2*y<0 && 3*x-2*y<=0, {x, y}]".
 */
private static String toMathematicaString(List<String> inequalities,
                                          String[] vars) {
    assert inequalities != null && !inequalities.isEmpty();
    assert vars != null && vars.length > 0;

    StringBuilder result = new StringBuilder("FindInstance[");

    for (String inequality : inequalities) {
        result.append(inequality);
        result.append(" && ");
    }
    // delete last " && "
    result.delete(result.length() - 4, result.length());

    result.append(", {"); // append variables
    for (String var : vars) {
        result.append(var);
        result.append(", ");
    }
    // delete last ", "
    result.delete(result.length() - 2, result.length());
    result.append("}]");

    return result.toString();
}
```

```
    // Mathematica's FindInstance returns "{}" iff the system is inconsistent.
    private static final String FIND_INSTANCE_ANSWER_IF_INCONSISTENT = "{}";

    private static boolean isInconsistentInMathematica(String problem) {
        String result = mathematicaLink.evaluateToOutputForm(problem, 0);
        return result.equals(FIND_INSTANCE_ANSWER_IF_INCONSISTENT);
    }
}
```

Поправки к диссертации, изложенной на с. 1–194

С. 12: в абзаце 4, строке 3 «посылкой» следует заменить на «посылок».

С. 39: третье предложение определения 2.2.1.1 (начинающееся со слова «Если») следует заменить на такой текст:
Если подчеркнутое одной чертой вхождение $\prec$ в формулу вида $F_1((G\underline{\preceq}H)\underline{\preceq}I)F_2$ является положительным (соответственно, отрицательным), то подчеркнутое двумя чертами вхождение $\prec$ в эту формулу является отрицательным (соответственно, положительным). Если подчеркнутое одной чертой вхождение $\prec$ в формулу вида $F_1(I\underline{\preceq}(G\underline{\preceq}H))F_2$ является положительным (соответственно, отрицательным), то подчеркнутое двумя чертами вхождение $\prec$ в эту формулу является положительным (соответственно, отрицательным).

С. 41: в правиле вывода $(\prec_+ q_n\forall)$ следует заменить «$q_n\forall$» на «$q_n \cdot \forall$».

С. 43: в правиле вывода $(\prec_- q_p\forall)$ следует заменить «$q_p\forall$» на «$q_p \cdot \forall$».

С. 52: из абзаца 3, строки 1 следует удалить предложение «Пусть посылка правила $(q_n\exists^-)$ общезначима.».

Следующие поправки на некоторых страницах из с. 102–114 исправляют одну погрешность: при контрприменении правила вывода все члены секвенции-заключения (для не минус-правила — кроме главного члена) переносятся без копирования в секвенции-посылки, поэтому число раскрытий того или иного вхождения квантора следует ассоциировать не просто с формулой (начинающейся с этого вхождения квантора), а с формулой и секвенцией, в которую эта формула входит.

С. 102: из диаграммы класса `Formula` следует удалить методы

```
public Object getLabel()
public void setLabel(Object lbl)
```

С. 103: требование *корректности разделения* следует изложить так (заменив два пункта вверху с. 103 на единственный пункт, приводимый здесь):

- объектные ссылки узлов, которые представляют любые два различные вхождения неатомарных формул в секвенцию (необязательно как членов секвенции), должны быть различны.

С. 103: перечень из двух пунктов внизу с. 103 следует дополнить, добавив слово «неатомарную» в самое начало первого пункта и добавив третий пункт:

- и вообще, вхождение неатомарной формулы в секвенцию среди других вхождений неатомарных формул в эту секвенцию.

С. 109: в строке 4 следует читать:

```
private LogicalSymbolFormula subformula
```

С. 110: строки 9–12 (кроме последнего слова строки 12) следует заменить на такой текст:

```
public Object getLabelFor(LogicalSymbolFormula formula);
public void setLabelFor(LogicalSymbolFormula formula,
                        Object label)
```

класса `Sequent`. Первый метод выдает пометку формулы в контексте данной секвенции, второй — помечает формулу (так же в контексте данной секвенции) объектом, указанным в качестве параметра.

С. 112: в вызовы методов `replaceVariable()`, `replaceSubformula()` и конструктора `Sequent()` следует добавить последним аргументом `sequent`.

С. 114: в заголовки методов `replaceVariable()` и `replaceSubformula()` следует добавить третий параметр — «`Sequent context`».

С. 160: в пункте [2] «Арнолд .К» следует заменить на «Арнолд К.».