Рогозин Олег Викторович, *кандидат технических наук, доцент кафедры ИУ-7 МГТУ им. Баумана, г. Москва*

Рогозин Николай Олегович, *ассистент кафедры ИУ-7, МГТУ им. Баумана, г. Москва*

АЛГОРИТМ ОБРАТНОГО ВЫВОДА ПРИ РЕШЕНИИ ЛОГИЧЕСКИХ ЗАДАЧ

В представленной статье рассматривается вариант реализации алгоритма обратного вывода при решении логических задач. Данный алгоритм широко применяется в системах искусственного интеллекта, экспертных системах. Приведён пример его использования в практических задачах. Подробно представлен сам алгоритм, а также грамматика для построения логических формул и высказываний.

Ключевые слова: *искусственный интеллект, алгоритм обратного вывода, экспертные системы.*

Введение

В построении современных вычислительных систем всё большее значение приобретают подходы и методы, основанные на использовании алгоритмов искусственного интеллекта. Начиная с восьмидесятых годов прошедшего столетия языки программирования с использованием элементов логического мышления получили широкое развитие. Так, в ЭВМ пятого поколения в качестве основного языка программирования было решено использовать язык программирования Пролог. В этом языке механизм логического вывода является встроенным и реализуется на основе алгоритма обратного вывода. В представленной статье рассматривается задача разработки модуля логического вывода по аналогии с реализованным в языке Пролог и демонстрируются возможности практического использования данного алгоритма.

Основные понятия, особенности и содержание задачи обратного вывода

Рассмотрим в качестве практического примера реализацию механизм обратного вывода в логической модели. Модель представления знаний будет содержать набор правил и фактов. Основные грамматические конструк-

ции модуля логического вывода представим на основе известной нотации Бэкуса — Наура:

<правило> ::= <заголовок правила> >
<список условий>
<список условий> ::= <условие> |
<список условий> & <условие>
<условие> ::= <предикат> (<список термов>)
<список термов> ::= <терм> |
<список термов>, <терм>
<терм> ::= <константа> | <переменная>
<факт> ::= <предикат> (<список констант>)
<список констант> ::= <константа> |
<список констант>,<константа>
<цель> ::= <предикат> (<список констант>)

Из представленной грамматики видно, что каждое правило может иметь несколько условий, соединённых логическим «И».

Рассмотрим алгоритм обратного вывода, который предусматривает движение по дереву вывода от цели, и представим его в «словесной форме».

Алгоритм обратного вывода

При обратном выводе, или поиске от цели, рассматривается цель, которой нужно достичь. Анализируются правила, ведущие к цели, и определяются условия их применения. Эти условия становятся новыми целями, или подцелями, поиска. Поиск продолжается в обратном направлении от достигнутых подцелей до тех пор, пока мы не достигнем исходных данных задачи. Таким образом определяется путь от данных к цели, который на самом деле строится в обратном направлении.

Алгоритм проверки верности цели

1. Проверяем, есть ли цель в списке фактов. Если есть, цель верна. Если нет, переход к п. 2.
2. Проверяем, можно ли заменить цель каким-либо фактом (предикатный символ факта совпадает с предикатным символом цели, количество термов одинаково, если в списке термов цели есть константы, они совпадают с соответствующими константами в списке термов факта). Если цель нельзя заменить фактом, переход к п. 3. Иначе — переход к п. 4.
3. Проверяем, можно ли вывести цель из какого-либо правила (предикатный символ заголовка правила совпадает с предикатным символом цели, количество термов в списке термов одинаково, если в списке термов заголовка правила есть константы, они совпадают с соответствующими константами в цели). Если цель нельзя вывести ни из какого правила, цель неверна. Если можно — переход к п. 3.
4. Унификация и получение новых подцелей.

При замене фактом — замена всех термов цели и всех текущих подцелей соответствующими константами найденного факта.

При выводе из правила — замена всех термов в заголовке найденного правила и этих же термов в условиях правила на соответствующие термы

цели. Новыми подцелями становятся условия правила.

При этом для того, чтобы исходная цель была верна, необходима истинность всех текущих подцелей. Если одна из подцелей заведомо неверна (см. п. 2 алгоритма), то необходимо вернуться на предыдущий шаг вывода. При этом поиск по подходящим правилам и фактам продолжается, начиная со следующих правил и фактов по отношению к тем, которые уже были рассмотрены. Если возвращение на предыдущий шаг невозможно, исходная цель неверна.

Рассмотрим работу нашего алгоритма на простом примере.

Пример

Правила:	**Факты:**
(1) f(x,y) > f(x,z) & f(z,y)	(1) f(1, 2)
(2) f(x,y) > f(y,x)	(2) f(2, 3)
Цель: (3) f(6, 3)	f(1, 6)

Решение:

1. Список целей: f(1, 6). В списке фактов нет такого факта, цель нельзя заменить никаким фактом, применяем правило (1).

2. Список подцелей: f(1, z), f(z, 6). Рассматриваем первую подцель.

Можно заменить фактом (1).

3. Список подцелей: f(1, 2), f(2, 6). Первая подцель верна. Ко второй подцели применяем правило (1).

4. Список подцелей: f(2, z), f(z, 6). Рассматриваем первую подцель.

Можно заменить фактом (2).

5. Список подцелей: f(2, 3), f(3, 6). Первая подцель верна. Ко второй подцели применяем правило (1).

6. Список подцелей: f(3, z), f(z, 6).

При неограниченном количестве рекурсий алгоритм зациклится, постоянно применяя правило (1) к новой полученной первой подцели. При ограничении числа рекурсий при возвращении к шагу 5 и применении ко второй подцели правила (2) получим подцель f(6, 3), которую на следующем шаге алгоритм признает верной. Все подцели верны, следовательно, верна исходная цель.

Программа решений в задаче определения предпочтений

Усложним решаемую проблему и проверим работу алгоритма в задаче определения предпочтений. Разработаем приложение, реализующее алгоритм обратного вывода. Для этого представим грамматику описания логических высказываний в виде следующих грамматических конструкций:

Элементарные строковые

константа ::= последовательность_строчных_букв
переменная ::= последовательность_заглавных_букв
отношение ::= последовательность_строчных_букв

Эти понятия представляют наименьшие логические элементы в строке, вводимой пользователем с клавиатуры. Понятие константа описывает постоянные значения, входящие в состав правил или фактов. Понятие переменная описывает переменные, входящие в состав правил или целевой формулы. Понятие отношение определяет формат строкового

значения для ввода отношений между субъектами в рамках фактов или правил.

Примеры:

константа: оля; шоколад; маша.
переменная: X; A; КТО.
отношение: любит; на; отец.
аргумент ::= константа | переменная

Понятие аргумент обобщает понятия константа и переменная и служит для описания аргументов в правиле базы знаний, которые могут быть как константами, так и переменными.
список_констант ::= константа | константа , список_констант
список_аргументов ::= аргумент | аргумент , список_аргументов

Эти понятия описывают соответствующие списки с символом запятой в качестве разделителя.

Предикатные формулы

утверждение ::= отношение (список_константа);
выражение ::= отношение (список_аргументов).

Эти понятия описывают простейший предикат — набор субъектов, связанных отношением. Только субъекты понятия утверждение задаются константами, а субъектами понятия выражения могут быть как константы, так и переменные.

Примеры:

утверждение: любит (оля, шоколад);
отец (вова, олег);
выражение: любит (лена, ЧТО);
отец (X, Y);
предикат ::= утверждение | выражение.

Понятие *предикат* обобщает понятия утверждение и выражение и служит для описания предикатов в правиле базы знаний, аргументы которых могут быть как константами, так и переменными.

набор_предикатов ::= предикат | предикат & набор_предикатов

Это понятие описывает набор предикатов с символом амперсанда в качестве разделителя.

формула ::= набор_предикатов

Понятие формула определяется как набор предикатов и служит для задания целевой формулы и условий правил базы знаний.

Объекты базы знаний

факт ::= утверждение.

Понятие *факт* служит для описания факта базы знаний и задаётся строкой утверждения, которая заканчивается точкой.

Пример:

любит (маша, конфеты).
правило ::= предикат < формула.

Понятие *правило* служит для описания правила базы знаний и состоит из двух частей. Заголовок (следствие) правила задаётся предикатом, а условия — формулой. Как и строка факта, строка правила заканчивается точкой.

Пример:

на (X, Z) < на (X, Y) & на (Y, Z).
цель ::= формула ?

Понятие *цель* служит для описания цели обратного вывода и задаёт-

ся целевой формулой. Строка цели завершается знаком вопроса.

Пример:

любит (маша, ЧТО) ?

Разработанная иерархия классов программы

Для проведения синтаксического анализа вводимых пользователем данных каждому ключевому понятию приведённой выше грамматики соответствует свой класс. Конструктор такого класса позволяет создать его экземпляр на основе текстовой строки, которая соответствует понятию грамматики. В случае ошибки конструктор класса генерирует исключение, соответствующее типу ошибки. Это исключение обрабатывается основной программой, реализующей пользовательский интерфейс, которая и выводит сообщение об ошибке.

Некоторые классы объединяют в себе несколько ключевых понятий грамматики, сходных между собой. Например, класс CValue может описывать понятия константа, переменная и отношение, а класс CPredicate — понятия утверждение и выражение. Одним из параметров конструктора таких классов обязательно является тип, задающий конкретное понятие.

На рис. 1 показана иерархия классов, а также направления включений и понятия грамматики, описываемые конкретными классами.

Классы, не описывающие понятий грамматики, служат для реализации синтаксического разбора и механизма обратного вывода. Класс CError описывает возможные ошибки и используется для генерации исключений. Класс CLink описывает набо-

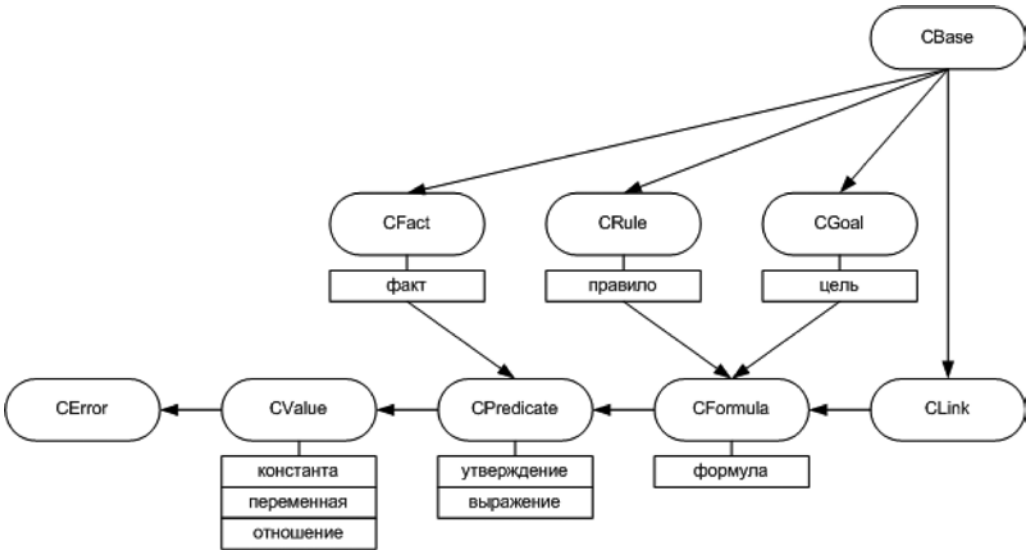


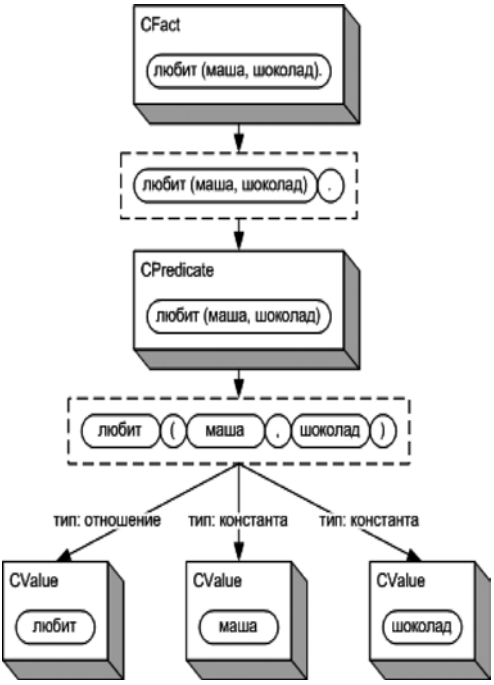
Рис. 1. *Иерархия классов и понятия грамматики, описываемые конкретными классами*

ры связанных переменных и служит для работы с ними в процессе обратного вывода. Класс CBase описывает всю базу знаний как набор фактов и правил, а также реализует механизм обратного вывода по заданной целевой формуле.

Синтаксический анализ

Конструктор каждого класса реализует синтаксический анализ строки, описывающей то конкретное понятие грамматики, которое соответствует данному классу. При этом он разбирает эту строку на составные части в соответствии с грамматическим определением данного понятия и вызывает конструкторы нижестоящих по иерархии классов, передавая им соответствующие части анализируемой строки.

Пример:



Как уже было сказано, на входе конструктора класса, описывающего несколько грамматических понятий, обязательно задаётся тип конструируемого экземпляра. Этот тип задаёт конкретное грамматическое понятие, позволяющее однозначно определить алгоритм синтаксического анализа полученной строки.

Механизм обратного вывода

Механизм обратного вывода основан на рекурсивной замене заданной целевой формулы в соответствии с фактами и правилами базы знаний. Алгоритм такого вывода выглядит следующим образом.

На первом шаге каждый предикат целевой формулы, содержащий переменные, подвергается возможной замене на соответствующий факт базы знаний. Новая полученная целевая формула рекурсивно анализируется по точно такому же алгоритму.

После того как все возможные подобные замены будут произведены, для каждого предиката целевой формулы ищется соответствующее правило базы знаний, согласно которому данный предикат можно заменить последовательностью других предикатов, составляющих условие правила. Новая формула также рекурсивно подвергается точно такой же процедуре, начиная с первого шага.

Таким образом, строится некоторое дерево вывода, которое обходится в соответствии со стратегией поиска в глубину. Целевая формула разбивается на части, которые заменяются новыми формулами, истинность которых

легче установить в рамках заданных фактов. Процесс продолжается до тех пор, пока целевая формула не превратится в аксиому, то есть все её предикаты не станут тождественно равными какому-либо факту базы знаний. Если же какой-либо предикат становится невозможно упростить, заменив его фактом либо применив к нему правило, то анализ такой целевой формулы также прекращается, так как она становится заведомо ложной.

Если исходная целевая формула содержала переменные, то любая замена предиката этой формулы сопровождается запоминанием соответствующих значений для каждой из этих переменных.

Высота дерева вывода ограничена глубиной рекурсии. В процессе работы алгоритма дерево вывода обходится целиком. При обходе листьев дерева, в которых целевая формула превращается в аксиому, происходит запоминание как самой формулы, так и множества значений переменных, о которых говорилось выше. Эта информация используется в качестве объяснения результатов вывода.

После того как обход дерева вывода будет полностью завершён, происходит оценка результата. Если за время вывода не было найдено ни одного преобразования, превращающего целевую формулу в аксиому, это означает, что данная формула ложна. Если же такие преобразования нашлись, то целевая формула истинна, а накопленная в процессе обхода дерева информация представляет собой совокупность объяснений.

Иными словами, если исходная целевая формула не содержала переменных, то объяснением результата

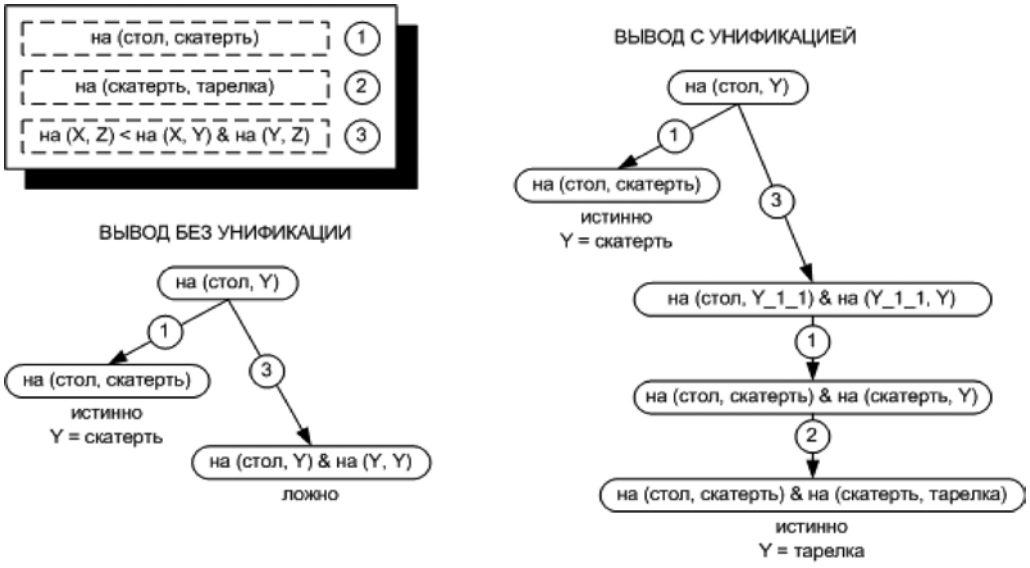


Рис. 2. *Необходимость унификации переменных правила в процессе обратного вывода*

вывода является тождественная ей целевая формула, превратившаяся в аксиому в результате преобразований. Если же в вопросе содержались переменные, то, в случае нескольких объяснений, к каждому из них добавляется реализация совокупности этих переменных.

Следует отметить, что при использовании правила для модификации целевой формулы все его переменные унифицируются путём добавления к ним номера правила и текущей глубины рекурсии (*рис. 2*). Это делается для того, чтобы имена связанных переменных не совпали с переменными исходной целевой формулы или другими промежуточными переменными. Использование знака подчёркивания делает невозможным для пользователя объявление переменной с таким же именем.

Примеры

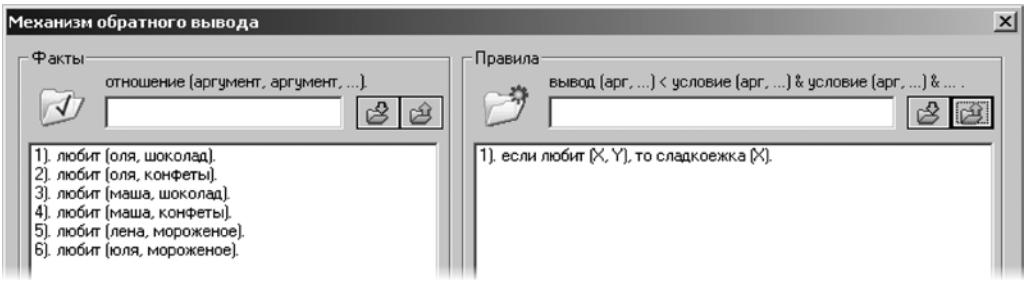
Пусть база данных содержит факты:

1) любит (оля, шоколад);
2) любит (оля, конфеты);
3) любит (маша, шоколад);
4) любит (маша, конфеты);
5) любит (лена, мороженое);
6) любит (юля, мороженое)

и правило:

7) сладкоежка (X) < любит (X, Y).

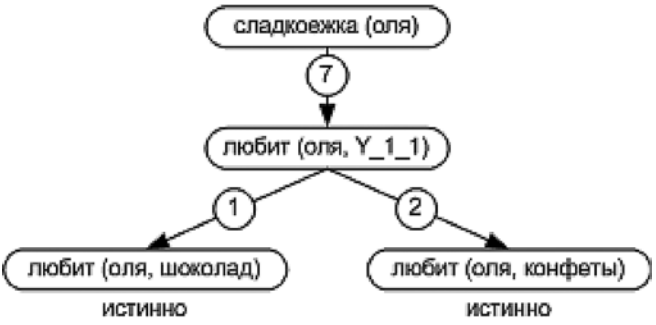
Фрагмент рабочего окна программы, содержащей данные знания:



ул

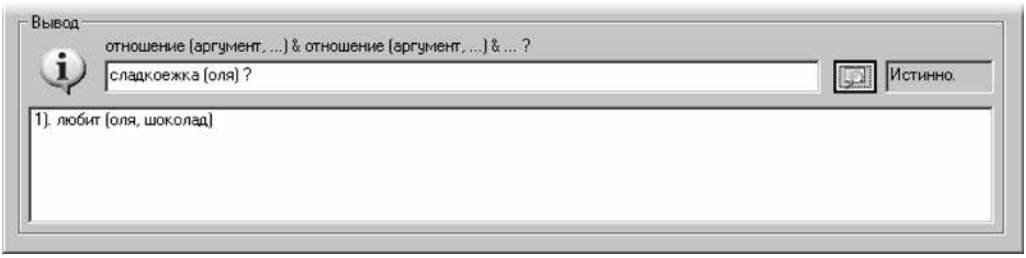
Пусть целевая формула: сладкоежка (оля) ?

Тогда дерево вывода будет выглядеть следующим образом:



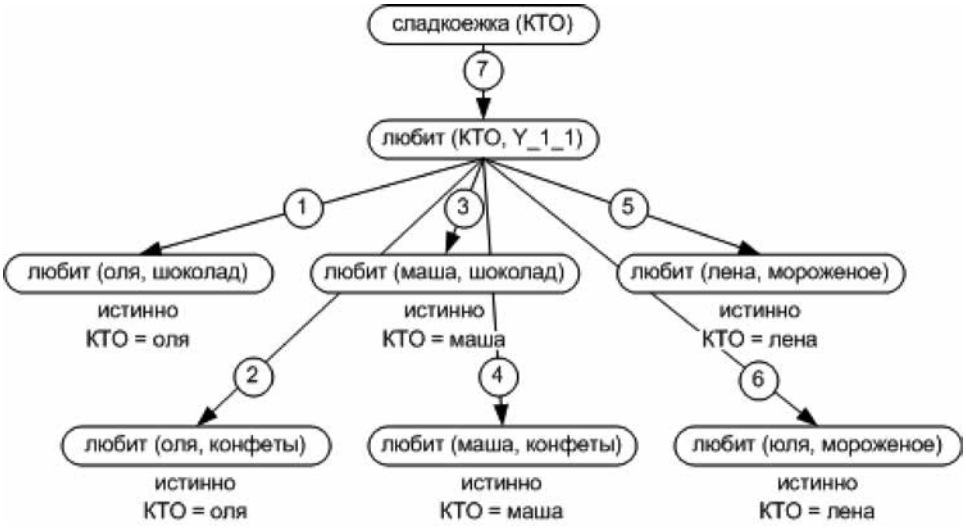
Так как целевая формула не содержала переменных, то, несмотря на то что в процессе обхода дерева вывода было найдено два преобразования, превращающих целевую формулу в аксиому, в качестве объяснения результата вывода используется только то, которое было получено первым.

Фрагмент рабочего окна программы, демонстрирующей данный пример:



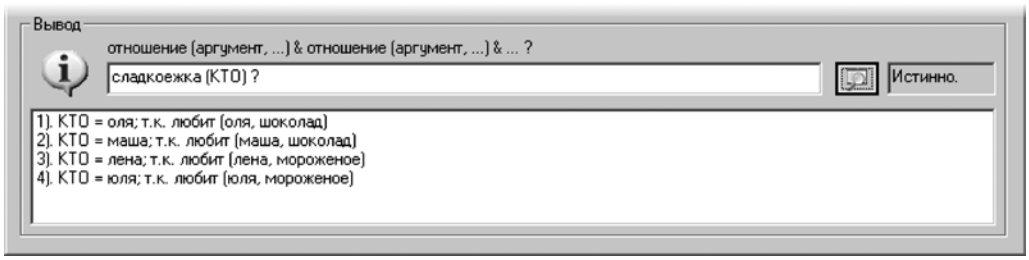
Теперь пусть целевая формула: сладкоежка (КТО) ?

Получим следующее дерево вывода:



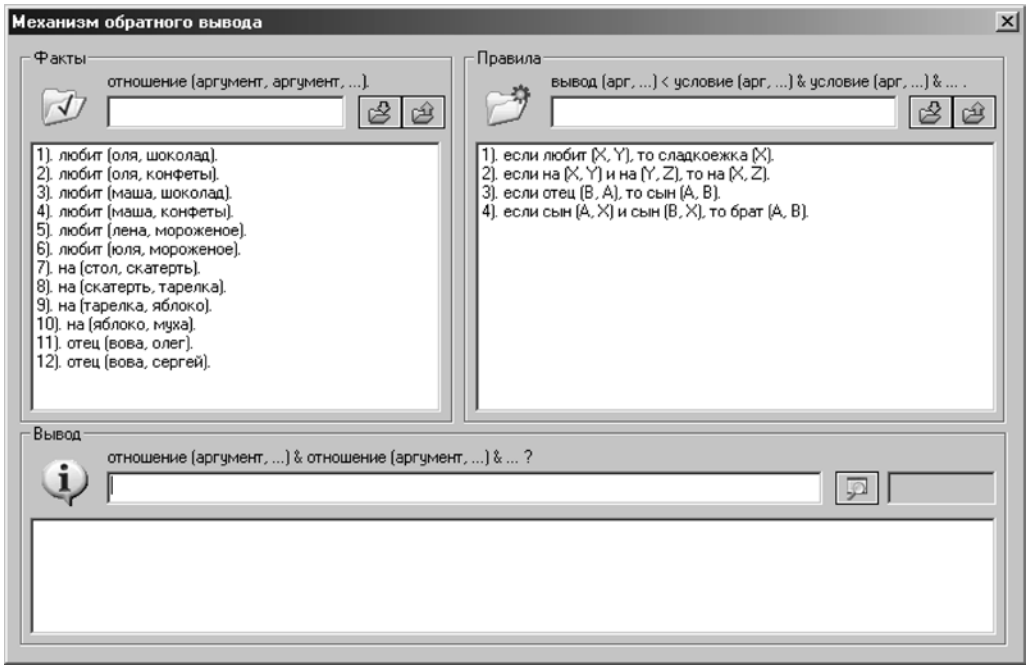
Следует отметить, что результатом вывода будут являться только *различные* реализации совокупности переменных исходной целевой формулы. То есть при получении нескольких преобразований, превращающих целевую формулу в аксиому при одних и тех же значениях исходных переменных, в качестве объяснения результата вывода используется только то преобразование, которое было получено первым.

Всё вышесказанное демонстрирует фрагмент рабочего окна программы, реализующей данный вывод:

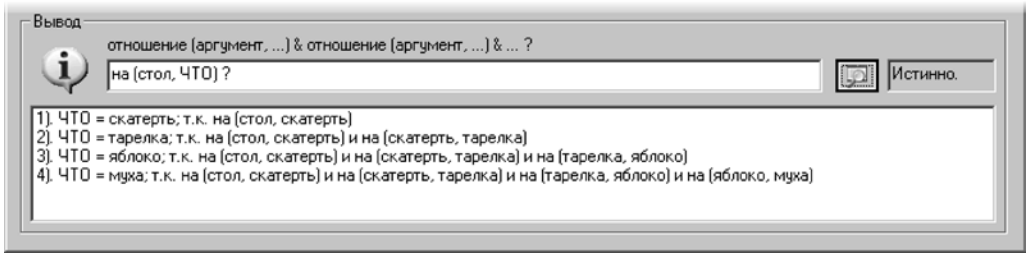


Тестовые примеры работы программы

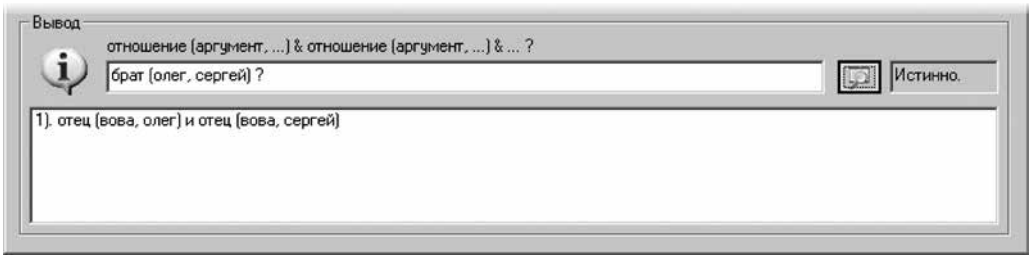
Исходное рабочее окно программы, содержащей тестовые примеры:



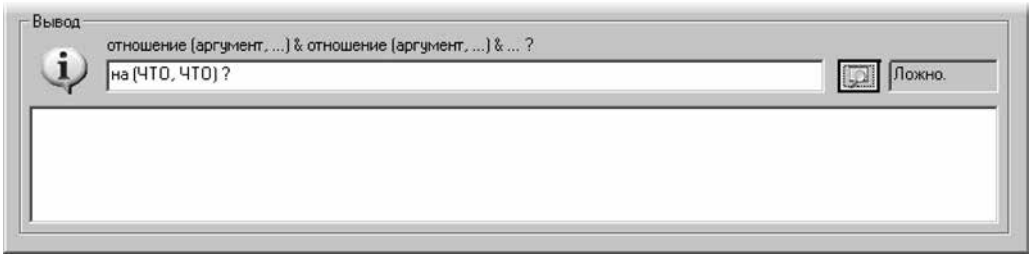
Демонстрация применения рекурсивных правил вывода:



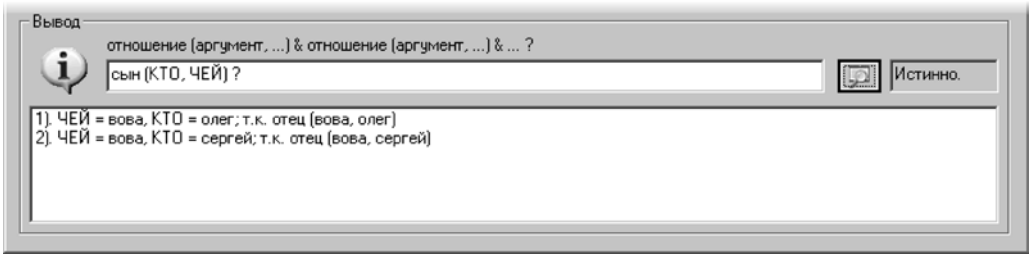
Демонстрация последовательного применения правил:



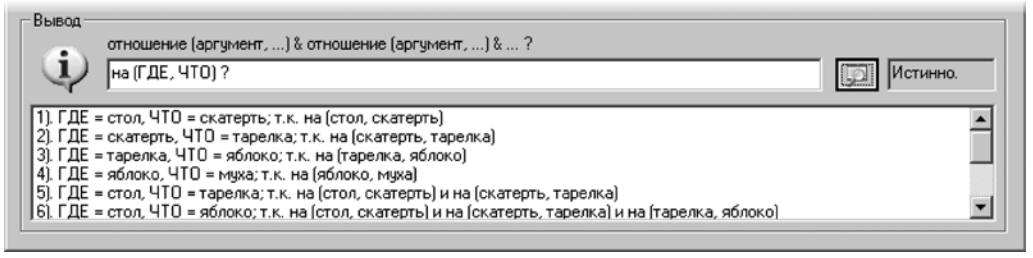
Демонстрация корректного связывания переменных:



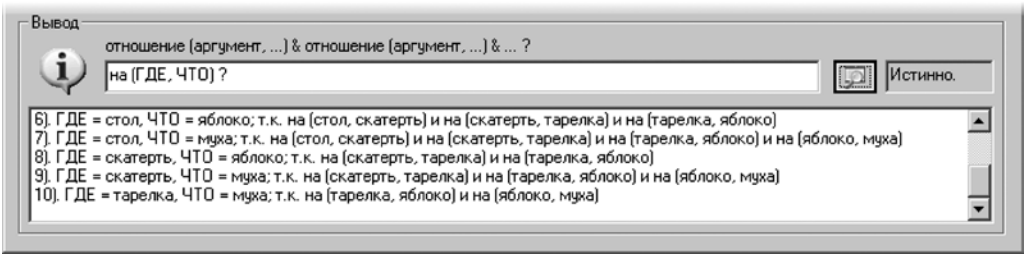
Демонстрация вывода по целевой формуле, содержащей несколько переменных:



Демонстрация вывода по целевой формуле, содержащей несколько переменных, с использованием рекурсивных правил:



Появившаяся полоса прокрутки позволяет посмотреть все остальные результаты, не поместившиеся в окне вывода:



ЛИТЕРАТУРА

1. *Волков И.К., Загоруйко Е.А.* Исследование операций: учеб. для вузов. — 2-е изд. / Под ред В.С. Зарубина, А.П. Крищенко. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2002.
2. *Мушик Э., Мюллер П.* Методы принятия технических решений. / пер. с нем. — М.: Мир, 1990.
3. *Орловский С.А.* Проблемы принятия решений при нечётких исходных данных. — М.:Наука,1981.
4. *Кини Р.Л., Райфа Х.* ПР при многих критериях предпочтения и замещения. — М.: Радио и связь, 1981.
5. *Борисов А.Н. и др.* ПР на основе нечётких моделей. Примеры использования. — Рига: Зинатне,1990.
6. *Солодовников И.В., Рогозин О.В., Шуруев О.В.* Реализация механизма логического вывода для прототипа экспертной системы (ЭС) Новые информационные технологии: материалы седьмого научно-практического семинара. — М.: Моск. гос. ин-т электроники и математики, 2004.
7. *Саати Т.* Принятие решений. Метод анализа иерархий. — М.: Сов. Радио,1993.
8. *Мелихов А.Н., Берштейн Л.С., Коровин С.Я.* Ситуационные советующие системы с нечёткой логикой. — М.: Наука,1990.
9. *Таха Х.* Введение в исследование операций: в2 т.. — М.: Мир,1985.
10. *Ермольев Ю.М., Ляшко И.И., Михалевич В.С., Тюптя В.И.* Математические методы исследования операций: учебн. пособие для вузов. — Киев: Вища школа. Головное изд-во, 1979.