Математические основы теории принятия решений

Олег Викторович Рогозин,
доцент кафедры «Программное обеспечение ЭВМ и информационные технологии» Московского государственного университета им. Н.Э. Баумана, кандидат технических наук

• *теория принятия решений* • *лица, принимающие решения (ЛПР)* • *математическая модель принятия решений* •

Основные понятия, особенности и содержание задачи

Альтернатива — это один из возможных способов достижения цели или один из конечных вариантов решений. Альтернативы отличаются друг от друга последовательностью и приёмами использования активных ресурсов. Для любой задачи принятия решений должна существовать тройка: цель, критерии, альтернативы. Если отсутствует один из компонентов, то нельзя говорить о постановке задачи. Присутствие менее чем одной альтернативы означает, что выбор отсутствует. Альтернативы характеризуются различными показателями их привлекательности для *лиц, принимающих решения* (далее ЛПР). Эти показатели называют признаками, атрибутами, критериями[1].

Критерий — это способ выражения различий в оценке альтернативных вариантов, с точки зрения участников процесса выбора, т.е. показатель привлекательности вариантов решений. Именно с помощью критерия ЛПР будет судить о предпочтительности исходов, а значит, и способов проведения операции по решению проблемы. Значимость того или иного из выбранных критериев определяется именно тем, что ЛПР не считает возможным выносить суждения о предпочтительности исхода операции, если именно того или иного критерия оценки недостаёт.

Схема процесса принятия решений

В классической книге лауреата нобелевской премии профессора Г. Саймона «The New Science of Management Decision» процесс принятия решения разбит на четыре основные фазы: сбор информации (intelligence); поиск и построение альтернатив (design); выбор альтернатив (choice); оценка результатов (review).

Первая фаза — сбор информации — сконцентрирована на идентификации проблемы принятия решения и сборе всей доступной информации о ней. При поиске и построении альтернатив (вторая фаза) центральным вопросом становится определение относительного небольшого числа альтернатив, которые следует изучить в деталях. На третьей фазе происходит выбор одного из вариантов решений из множества альтернатив, подготовленных на второй фазе. Последний шаг в процессе принятия решений — это реализация выбранной альтернативы и обобщение опыта, полученного в процессе решения проблемы.

Таким образом, само решение принимается в рамках второй и третьей фаз, включающих в себя:
— конструирование относительно небольшого множества альтернатив;
— окончательный выбор варианта решения из сформированного множества.

Функция выбора в теории принятия решения имеет фундаментальное значение. Именно на её построение в конечном итоге ориентированы решение задач формирования ис-

[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.

ходного множества альтернатив, анализ условий проведения операций, выявление и измерение предпочтений лица, принимающего решение.

Этапы процесса принятия решения

Рассмотрим схему процесса принятия решений, представленную на рис. 1.

Основу принятия всех решений на всех этапах процесса выработки решений составляют предпочтения ЛПР. В качестве целесообразного начала принятия решений примем формализацию предпочтений. Когда ЛПР должно выбрать какое-то из нескольких альтернативных действий, оно обычно стремится выбрать «наилучшее» из них. При построении математической модели принятия решений предпочтения ЛПР, как правило, описываются введённой априори целевой функцией *f*, значения которой *f*(a) для данного допустимого действия представляет его полезность для ЛПР.

После того как предпочтения ЛПР формализованы с требуемым качеством, а также получена необходимая информация о предпочтениях, переходят к следующему важному шагу принятия решений — к построению функции выбора.

1. ПОСТРОЕНИЕ ФУНКЦИИ ВЫБОРА

В условиях определенности:

- по скалярному критерию;
- по векторному критерию.

В условиях неопределенности :

- стохастической;
- поведенческой;
- природной;

2. ОТЫСКАНИЕ РАЦИОНАЛЬНЫХ АЛЬТЕРНАТИВ

- Содержательный анализ рациональных альтернатив (интеграция и адаптация к особенностям реальной проблемной ситуации).

3. ВЫБОР НАИЛУЧШЕГО РЕШЕНИЯ ДЛЯ РЕАЛИЗАЦИИ

- Разработка плана и реализация принятого решения.
- Оценка фактически достигнутых результатов

Рис. 1. Схема процесса принятия решений

Во многих практических задачах (выбор капиталовложений в рамках фирмы; управление кадрами — подбор и расстановка; задачи упорядочения и согласования) сопоставление действий между собой должно быть проведено с учётом многочисленных разнородных последствий. Очевидно, что трудно выразить обобщённую оценку действия с учётом всех его последствий в виде единственного числа. В чём же заключается задача исследователя, ставящего своей целью помочь ЛПР при выборе среди имеющихся альтернатив?

Можно выделить следующие четыре основных подхода:

1) объединение (агрегирование) многих целевых функций в единую функцию, позволяющую полностью упорядочить рассматриваемое множество альтернатив по их предпочтительности;

2) последовательное выявление предпочтений одновременно с исследованием допустимого множества альтернатив;

3) нахождение для имеющихся альтернатив $a \in A$ пусть не полного, а лишь частичного упорядочения, но более информативного, чем просто объединение не противоречащих друг другу предпочтений, устанавливаемых в соответствии с каждой из n привлекаемых целевых функций $f_i(a)$, $i = 1,2, \ldots, n$;

4) максимально возможное уменьшение неопределённости и несравнимости.

Классификация задач принятия решений по виду отображения множества допустимых альтернатив

Представим задачу принятия решений в виде следующего набора:

{T, A, X, F, G, D}

где T — постановка задачи;
A — множество допустимых альтернативных вариантов;

Грамматика

Для обеспечения ввода пользователем правил и фактов в базу знаний была разработана следующая грамматика.

Элементарные строковые значения

```
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_констант)

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 задаётся целевой формулой. Строка цели завершается знаком вопроса.

Пример: любит (маша, ЧТО)?

Иерархия классов

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. 2 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 наборы 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.

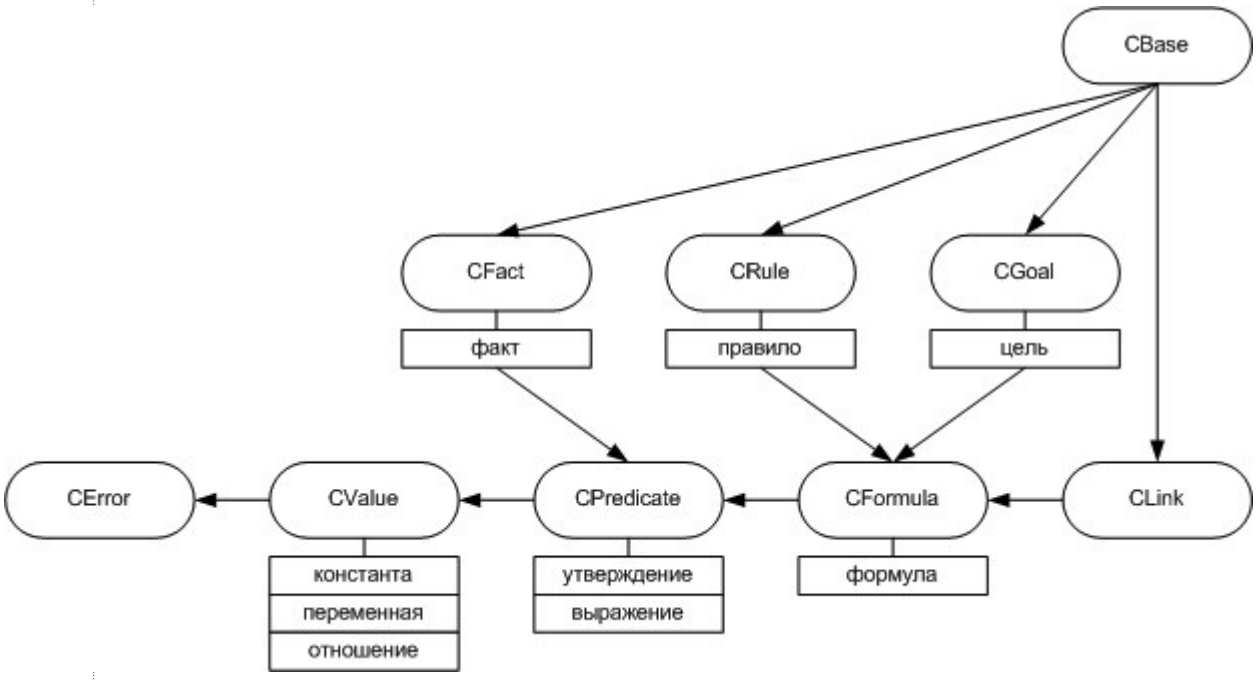
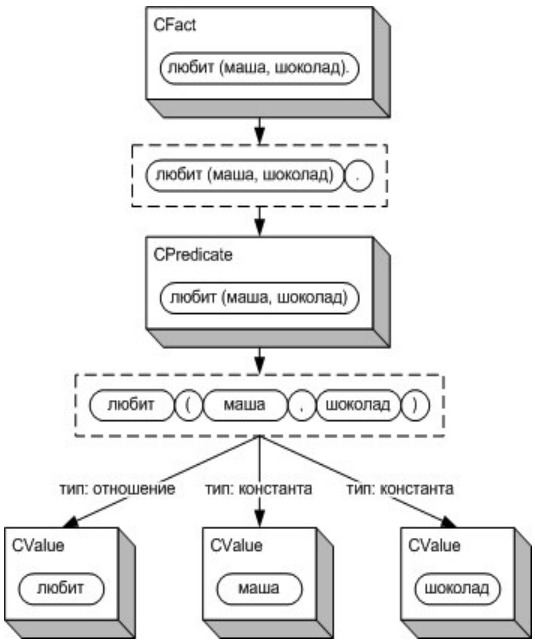


Рис. 2. Иерархия классов и понятия грамматики, описываемые конкретными классами

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. Как упоминалось выше, при возникновении ошибки на любом этапе генерируется исключение, которое сразу же обрабатывается основной программой.

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, заменяющиеся 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, значит, эта 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 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 их 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. 3). 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:

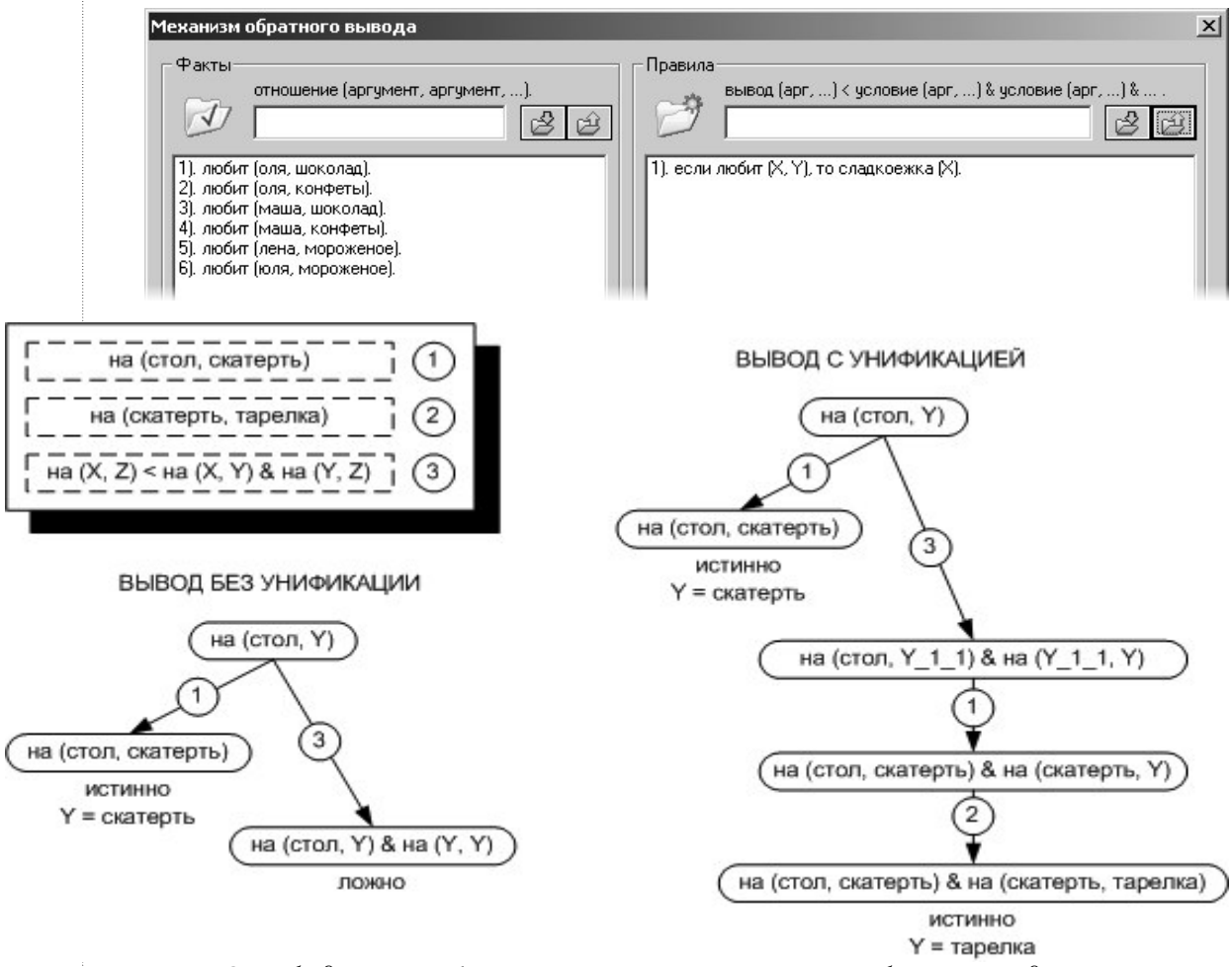
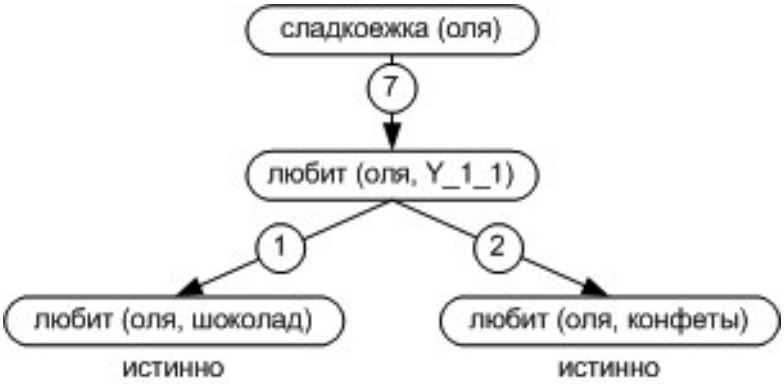


Рис. 3. Необходимость унификации переменных правила в процессе обратного вывода

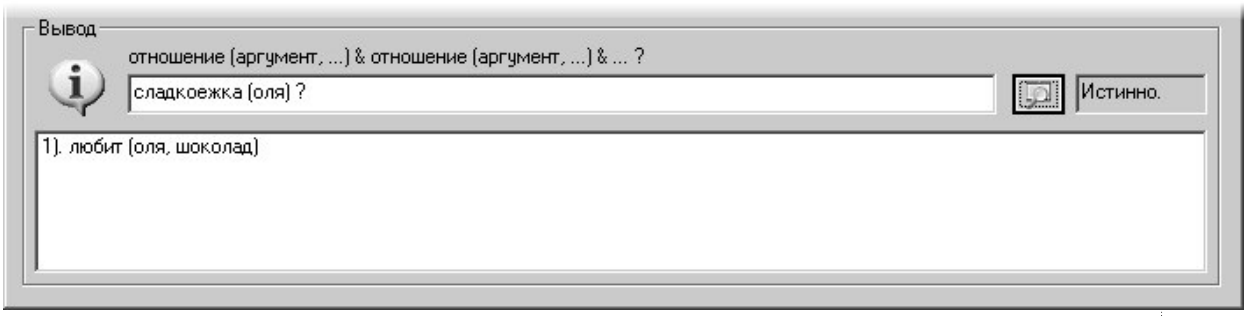
Пусть целевая формула: сладкоежка (Оля).

Тогда дерево вывода будет выглядеть следующим образом:



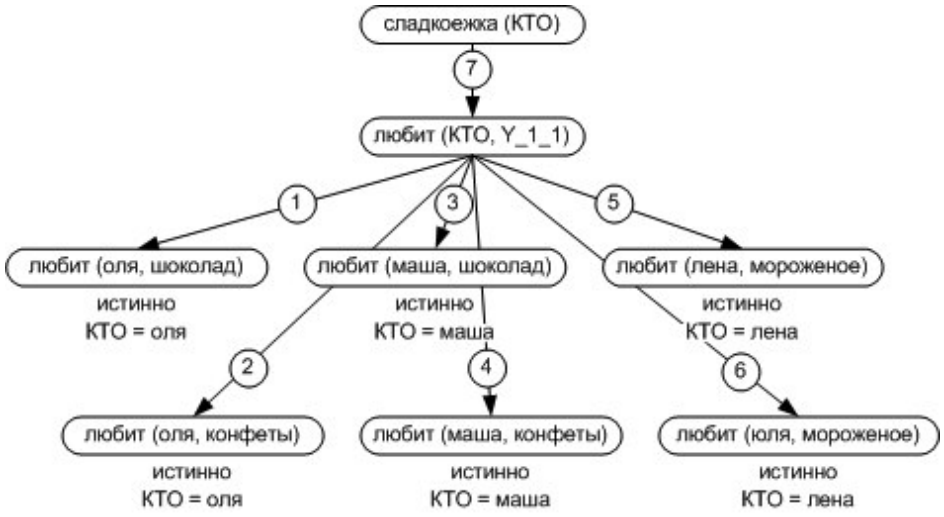
Так как целевая формула не содержала переменных, то, несмотря на то, что в процессе обхода дерева вывода было найдено два преобразования, превращающих целевую формулу в аксиому, в качестве объяснения результата вывода используется только то, которое было получено первым.

Фрагмент рабочего окна программы, демонстрирующей данный пример:



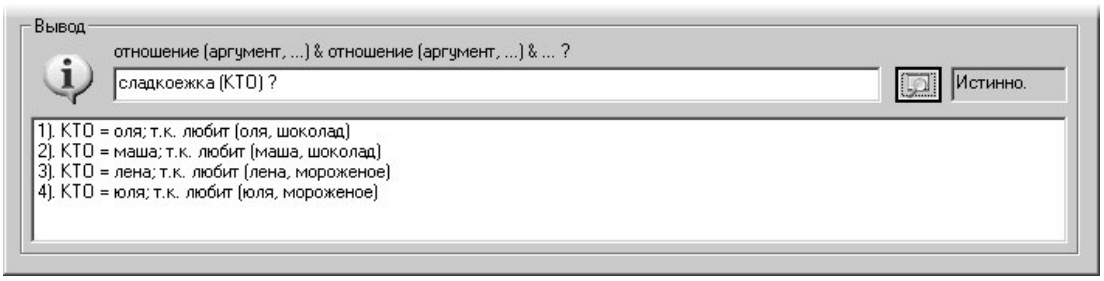
Теперь пусть целевая формула: сладкоежка (КТО).

Получим следующее дерево вывода:



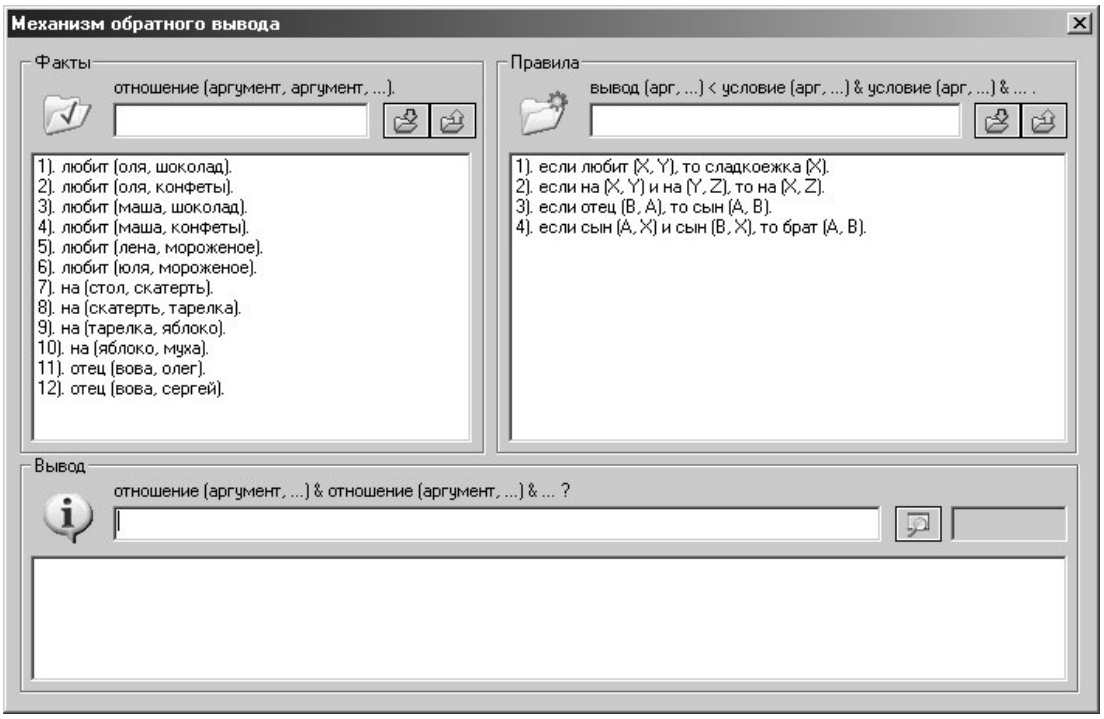
Следует отметить, что результатом вывода будут являться только *различные* реализации совокупности переменных исходной целевой формулы. То есть при получении нескольких преобразований, превращающих целевую формулу в аксиому при одних и тех же значениях исходных переменных, в качестве объяснения результата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

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

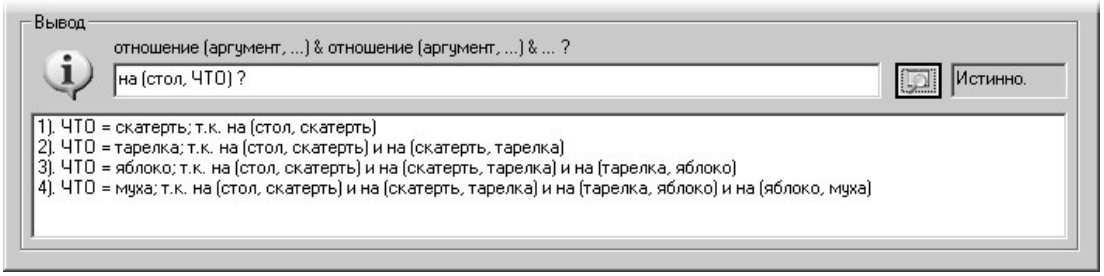


Тестовые примеры работы программы

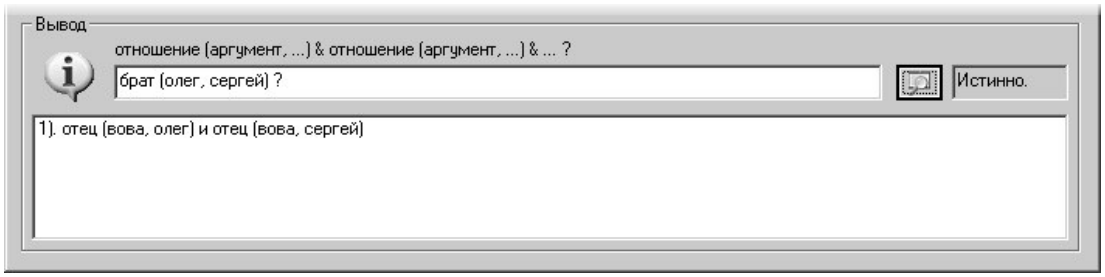
Исходное рабочее окно программы, содержащей тестовые примеры:



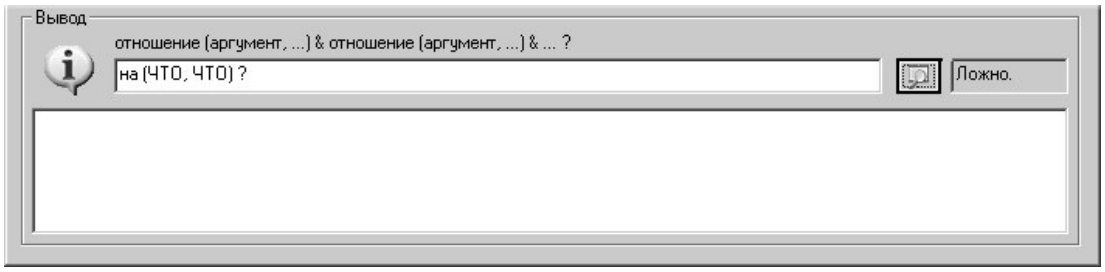
Демонстрация применения рекурсивных правил вывода:



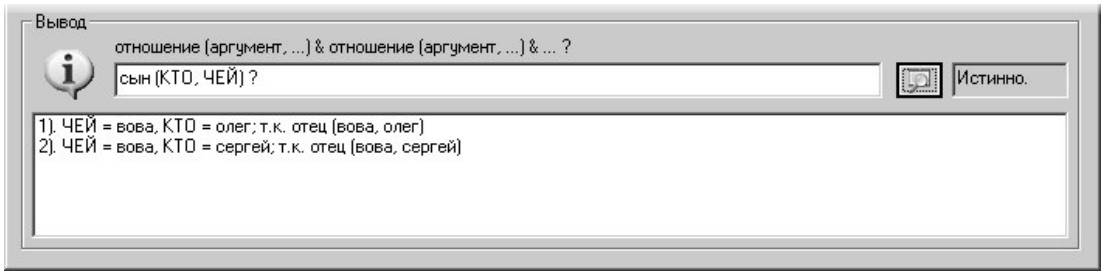
Демонстрация последовательного применения правил:



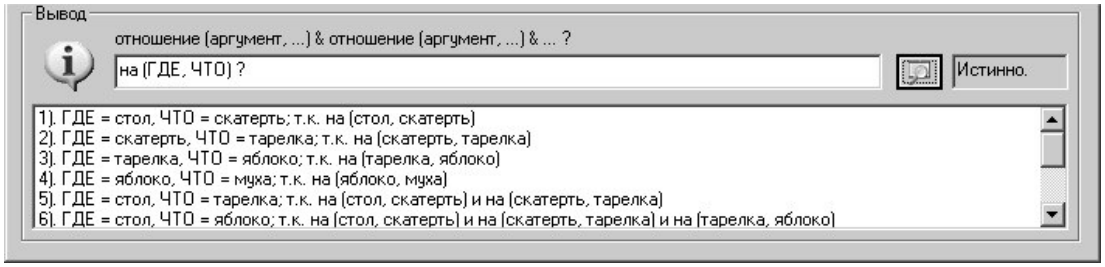
Демонстрация корректного связывания переменных:



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: