МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Белорусский национальный технический университет

Кафедра «Системы автоматизированного проектирования»

О.В. Герман
Ю.О. Герман

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Методическое пособие

Минск
БНТУ
2013

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Системы автоматизированного проектирования»

О.В. Герман
Ю.О. Герман

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Методическое пособие
для студентов специализации 1-40 01 02-01 «Информационные системы и технологии в проектировании и производстве»

Минск
БНТУ
2013

УДК 004.8(075.8)
ББК 32.813я7
Г38

Р е ц е н з е н т ы:
канд. техн. наук, доцент кафедры ИСИТ БГТУ, *Н.И. Гурин*;
д-р техн. наук, профессор кафедры ИТАС БГУИР, *В.С. Муха*

Герман, О.В.

Г38 Искусственный интеллект: методическое пособие для студентов специализации 1-40 01 02-01 «Информационные системы и технологии в проектировании и производстве» / О.В. Герман, Ю.О. Герман. – Минск: БНТУ, 2013. – 127 с.

ISBN 978-985-525-750-0.

Настоящий материал предназначен для использования в качестве методического пособия при изучении дисциплины «Искусственный интеллект». Пособие содержит сведения, которые можно использовать в качестве дополнения и расширения лекционного курса, а также как реферативный источник. Представлены важнейшие темы современного теоретического курса по проблематике искусственного интеллекта с ориентацией на практическое применение. Содержит шестнадцать лекций и список литературы. Материал соответствует учебной программе.

УДК 004.8(075.8)
ББК 32.813я7

ISBN 978-985-525-750-0

СОДЕРЖАНИЕ

ЛЕКЦИЯ 1

Основные понятия искусственного интеллекта и интеллектуальных информационных систем. Прикладные системы искусственного интеллекта

Искусственный интеллект – это научное и прикладное направление, объединяющее группу теорий и методологий, предназначенных для решения *интеллектуальных задач*. Понятие интеллектуальной задачи можно считать первичным. Интеллектуальные задачи – это особый тип задач наряду с другими известными задачами. Они характеризуются своими *специфическими чертами*:

- задача является не полностью определенной;
- критерий, как правило, отсутствует;
- нет “хорошего” алгоритма решения, т.е. либо алгоритм является переборным, либо алгоритм не известен вовсе;
- задача характеризуется значительным пространством поиска;
- имеются эксперты по рассматриваемой проблематике.

Следует заметить, что для отнесения задачи к категории “интеллектуальных” не обязательно одновременное выполнение всех указанных условий.

Частичная определенность задачи связана с недостаточными знаниями о предметной области. Для некоторых задач это свойство является определяющим. Сама задача построения логического вывода, а также многие оптимизационные задачи комбинаторной и дискретной математики обладают свойством частичной определенности, поскольку не для всякой формулы разрешим вопрос о ее истинности.

Для интеллектуальных задач, как правило, нет или не известно хороших алгоритмов решения. Например, не известно, как искать выигрышное продолжение в шахматной партии (кроме перебора вариантов, которых огромное множество), как разместить предметы различной формы в заданной прямоугольной или иной области, как искать целочисленные корни произвольного многочлена от нескольких переменных и т.д. Решение логических задач и построение доказательств также относится к категории интеллектуальных задач.

Исходя из указанных особенностей можно считать интеллектуальными многие задачи – постановка диагноза в медицине и технике, прогнозирование исходов событий особенно при отсутствии статистических данных (например, угадывание исхода соревнования), распознавание образов, речи и текста (особенно рукописного), синтез программ (задается что известно и что требуется определить; машина должна сама построить программу для решения), многие комбинаторные задачи на графах – например, отыскание минимального покрытия графа множеством вершин, отыскание гамильтонова цикла, задача о минимальной раскраске вершин графа и пр., игровые задачи типа шахмат и карт, планирование поведения роботов и т.д.

Задачу *P* можно по форме представить таким образом [1, 2]:

P =<Модель, Исходное_состояние, Критерий, Решение,
Решающая процедура, Доказательство>

Модель – это множество формул, описывающих связи в предметной области задачи. В этом смысле модель можно трактовать как базу знаний. **Исходное состояние** – это то, что известно о значениях объектов (переменных) задачи. **Критерий** – это условие, которое должно быть обеспечено решением задачи. **Решение** – это набор значений неизвестных переменных задачи. **Доказательство** – это математическое формальное обоснование правильности решения.

Обычно в задаче требуется найти Решение и Решающую процедуру. Однако имеются задачи, требующие найти исходное состояние, удовлетворяющее решению, и даже критерий. Весьма сложной категорией задач являются задачи, требующие найти решающую процедуру. Эта универсальная проблема, как доказали математики, не имеет алгоритмического решения. Иначе говоря, нет алгоритма для создания алгоритмов решения задач. Вместе с тем указанный факт вовсе не очевиден. Еще великие Лейбниц и Гильберт рассчитывали такой универсальный алгоритм построить.

Важной особенностью любой технологии, связанной с решением интеллектуальных задач, является использование **слабых методов,** к которым относятся:

- ограниченный и направленный перебор;
- исключение и отсечение;
- эвристики;
- индукция.

Основной вопрос применения слабых методов поиска решения состоит в том, чтобы на их основе построить точный или статистически точный метод. Этот момент является важнейшим. Проведем аналогию со стрельбой по цели. Если вероятность поражения мишени не равна 1, то нужно стрелять много раз. Таким образом, слабый метод должен применяться многократно. При этом каждое очередное его применение должно осуществляться по новой траектории (нельзя направлять пули в оду и ту же точку). Траектория должна сходиться к цели. Для этого слабый метод должен сокращать пространство поиска, отсекая целые области, где решения быть не может (принцип отсечений). Третье: нужен критерий завершения стрельбы. Как правило, такой критерий имеет вероятностную природу. Таким образом, именно идея использования слабых методов может дать решение «проблемы размерности», присущей точным методам (см. характеристику *класса 1* задач ИИ, помещенную далее по тексту).

В настоящее время **сфер применения** систем искусственного интеллекта достаточно много. Речь идет о разработке интеллектуальных информационных систем в той или иной области. Прежде всего – это различные экспертные системы (ЭС) в медицине, проектировании, военном деле, геологоразведке, машиностроении, банковской сфере и т.д. Например, задача выделения кредитов связана с риском не возврата или несвоевременного возврата и требует оценки надежности клиента по множеству критериев, каждый из которых лишь частично характеризует финансовую благонадежность клиента. Аналогично, вложение денег в рискованные, но сулящие большие доходы сферы. Например, хорошо известно, что много денег вложили в Интернет в связи с бумом интернет-технологий, однако значительная часть проектов оказалась убыточной. В настоящее время весьма актуальна задача поиска ответов на смысловые вопросы (в том же Интернете). Смысловой вопрос требует точечного (точного) ответа, а не указания множества ссылок на документы, как это делается в поисковиках типа Google.

В экспертных системах велика роль **человека-решателя**. В настоящее время это обусловлено его незаменимостью в части генерации гипотез и обработки неформализованных знаний. Технология решения задач человеком отличается от машинной арифметики. Интуитивная составляющая механизма угадывания имеет тем большее значение, чем менее формализована задача.

Таким образом, представление задачи играет важную роль в плане разделения ролей человека и машины в процессе решения. Общую парадигму ЭС с учетом сказанного можно представить следующим образом:

ЭС= База Знаний + Машина Вывода + Человек-решатель (1.1)

Эту же парадигму можно применить и к произвольной интеллектуальной информационной системе (ИИС), расширив ее с учетом функциональной специфики:

ИИС= База Знаний (база данных) + Машина Вывода +
Человек-решатель+Прикладное_обеспечение (1.2)

Прикладное обеспечение может включать систему машинной графики, статистической обработки, формирования отчетности, САПР и пр.

Для построения ИИС нужны:

1. математические методы для обработки знаний и модели для представления знаний;
2. программные средства для работы со знаниями.

В настоящее время имеется достаточно широкий спектр **математических методов** для работы со знаниями: методы распознавания образов, методы классической математической логики и неклассических логик, статистические методы, методы поиска решений на основе эвристических механизмов на деревьях состояний, методы принятия решений, использование нейросетевых моделей и др.

Имеется спектр программных систем для работы со знаниями. Отметим здесь в первую очередь языки искусственного интеллекта типа Пролога и Лиспа, прикладные экспертные системы типа Project Expert, EMYCIN (известная медицинская экспертная система), GURU (оболочковая экспертная система широкого назначения), различные прикладные пакеты, например, для работы с нейросетями и т.п.

Таким образом, констатируем следующее.

Имеется набор специфических задач, которые можно назвать интеллектуальными. Для решения этих задач разработан широкий набор математических методов. Заметим, что одного какого-то метода нет, поскольку интеллектуальные задачи сложные и требуют многостороннего рассмотрения под различными углами зрения. Вместе с тем можно выделить два главных направления в создании методов решения интеллектуальных задач.

Первое. Имитация и подражание естественным механизмам мозга и живой природы. Это направление включает нейросети, генетические алгоритмы, механизмы эвристического поиска.

Второе. Методы, группирующиеся вокруг математической логики (как классической) так и неклассических.

Математическая логика работает с объектами произвольной природы и отношениями (предикатами), описывающими связи между объектами. Следовательно, по своему духу математическая логика ориентирована на обработку и представление знаний в наибольшей степени, чем любая иная математическая дисциплина, например та же математическая статистика.

Наконец, имеется набор программных средств и технологий для работы со знаниями, позволяющих практически воплотить идеи искусственного интеллекта.

Рассмотрим более детально связь методов решения задач в ИИС (ЭС с классами задач, на которые они ориентированы.

Класс 1: малое пространство поиска с надежными знаниями и данными. Для этого класса задач основным методом решения является исчерпывающий перебор или метод отсечений неподходящих вариантов. При исчерпывающем поиске максимальный размер пространства поиска зависит от времени, необходимого на просмотр

одного решения. Если на просмотр одного решения тратится 25 микросекунд, то на 10! решений потребуется 24 часа. Такой размер пространства часто соответствует верхнему пределу для реальных задач при полном перебое.

Класс 2: малое пространство поиска с ненадежными знаниями. В этом случае используются методы последовательного анализа вариантов на диаграмме состояния задачи. Стержнем методов является теорема Байеса, другие вероятностные подходы, например, метод Вальда.

Класс 3: модель задачи изменяется со временем. Например, задачи планирования поведения роботов и предсказания требуют рассуждений о всех возможных будущих мирах и событиях. Методы этой группы образуют так называемое ситуационное исчисление, временную логику и т.п.

Класс 4: большое пространство состояний. Основным подходом к решению задач такого типа является механизм отсечений, использующий порождение гипотез и их проверку. Ключевым моментом является выполнение отсечений на наиболее ранних этапах построения решения, что сокращает перебор и обеспечивает повышение эффективности поиска решения. Механизм отсечений реализован в методе ветвей и границ, принципе сжимающих отображений и др.

Класс 5: задачи, требующие выдвижения гипотез и правдоподобные рассуждения. Методы, используемые для решения этих задач, основаны на различных неклассических логиках: вероятностной, нечеткой, логике возможностей и др.

ЛЕКЦИЯ 2

Модели знаний: логические, семантические сети, фреймы, продукционные модели

Имеется несколько хорошо разработанных моделей знаний, обладающих своими достоинствами и недостатками:

- логические модели;
- продукционные модели;
- фреймы;
- семантические сети.

Каждая из этих моделей допускает также деление. Например, в классе логических моделей имеются модели на основе логики высказываний и логики предикатов. Различают также модели на основе четкой и нечеткой логики, двузначной и многозначной, противоречивой и непротиворечивой логики и т.д.

Логическая модель знаний

Логические модели наиболее универсальны и формализованы. Для них имеется мощный математический аппарат. Логическая модель представляет конечное или бесконечное множество логических формул, называемых предикатами. Для записи формул используется язык, содержащий алфавит (набор символов) и множество правил образования правильно построенных формул. Символами логического языка являются:

- константы (например, 0,1,2,… или ‘a’,’b’,’c’,…);
- переменные: x,y,…,z;
- функциональные символы: f,g,h, … (функциональные символы называются также функторами);
- символы предикатов (отношений) P,Q,R,…
- кванторы всеобщности $\forall$ и существования $\exists$;
- логические операции (связки): & (“и”), v (“или”), $\neg$ (“не”), $\rightarrow$ (“следует”), $\leftrightarrow$ (“эквивалентно”). & называют операцией конъюнкции, v – дизъюнкции, $\neg$ – отрицания, $\rightarrow$ – импликацией и $\leftrightarrow$ – эквиваленцией.

Кроме символов языка, рассматриваются синтаксические правила записи предложений (формул) логического языка. Отправным элементом является простейшая формула – **предикат** (в логике предикатов) и булевская переменная (в логике высказываний).

Предикат – это простейшая (атомарная) формула (с отрицанием или без него) вида P(…), где в скобках указываются аргументы формулы – переменные, константы или функторы, либо вовсе не указываются аргументы. Предикат принимает одно из двух возможных значений: истина или ложь. Аргументы предиката имеют произвольную природу. Каждый предикат есть формула логического языка. Пусть P, Q – любые две формулы. Тогда формулами также являются следующие выражения [3]:

1. $\neg P, \neg Q;$
2. $P \vee Q;$
3. $P \,\&\, Q;$
4. $P \rightarrow Q;$ (2.1)
5. $P \leftrightarrow Q;$
6. $\forall x\, P(x);$
7. $\exists x\, P(x)$

В дальнейшем знак &, как правило, будем опускать, а знак $\leftrightarrow$ заменять на = или $\equiv$.

Определение. 1. Дизъюнктом (в логике высказываний) называется дизъюнкция литер (булевских переменных) с отрицанием или без, например, $x \vee y \vee \neg z$. 2. Дизъюнктом в логике предикатов называется дизъюнкция литералов (атомарных формул), взятых с отрицанием или без него, например, $P(a,z) \vee \neg Q(z,f(x))$.

Определение. Конъюнктивной нормальной формой (КНФ) называется конъюнкция дизъюнктов.

Определение. Выполняющей интерпретацией I для заданной КНФ называется множество значений переменных этой КНФ, при которых она истинна. Например, КНФ $(x_1 \vee \neg x_2)(\neg x_1 \vee x_2)(\neg x_2 \vee \neg x_3)(x_1 \vee x_3)$ истинная в интерпретации $I=<x_1=1,x_2=1,x_3=0>$ либо в интерпретации $<x_1=0, x_2=0, x_3=1>$.

Число различных интерпретаций для дизъюнкта с $n>0$ переменными равно 2^n. Однако не все они являются в общем случае выполняющими интерпретациями.

Определение. Задача ВЫПОЛНИМОСТЬ (коротко – ВЫП) формулируется так: для данной КНФ установить, имеется ли для нее хотя бы одна выполняющая интерпретация.

Определение. Формула B следует из формулы A, если в любой интерпретации, где A истинна, B также истинна. Пишем $A \rightarrow B$ в этом случае.

Определение. Формула A тождественно истинна, если она истинна в любой интерпретации. Тождественно истинная формула называется также тавтологией.

Рассмотрим примеры построения логических моделей.

Пример1. В хищении подозреваются A, B и C. Известно следующее.

1. Никто, кроме A,B,C в хищении не замешан.
2. A никогда не ходит на дело без соучастников.
3. C не виновен, если виновен B.

Спрашивается, виновен ли A?

Нужно составить систему логических уравнений, введя необходимые булевские переменные. Система переменных должна быть адекватна условиям задачи. Обозначим:

x_a – виновен A, x_b – виновен B, x_c – виновен C. Составим следующую базу знаний:

1. $x_a \vee x_b \vee x_c$
2. $x_a \rightarrow x_b \vee x_c$
3. $x_b \rightarrow \neg x_c$

Эти логические формулы полностью соответствуют условиям задачи.Относительно этой модели знаний поставлен вопрос: верно ли что $x_a = 1$? Ответ на этот вопрос должна дать **машина вывода**.

Пример 2. Следует назначить по одной работе w1, w2, w3, w4 четырем исполнителям a,b,c,d, если известно, что a может выполнить либо работу w2, либо работу w3; b может выполнить

работу w1 или w2; c может выполнить w1 или w3 или w4; d может выполнить работу w2 или w4.

Базу знаний для этой задачи можно записать с помощью формул логики предикатов таким образом:

//группа условий, показывающих, какие работы могут быть назначены исполнителям

$P(a,X) \rightarrow (X=w2) \vee (X=w3)$
$P(b,Y) \rightarrow (Y=w1) \vee (Y=w2)$
$P(c,Z) \rightarrow (Z=w1) \vee (Z=w3) \vee (Z=w4)$
$P(d,R) \rightarrow (R=w2) \vee (R=w4)$

//группа условий, показывающих, какие исполнители могут быть назначены на работы

$P(X1,w1) \rightarrow (X1=b) \vee (X1=c)$
$P(Y1,w2) \rightarrow (Y1=a) \vee (Y1=b) \vee (Y1=d)$
$P(Z1,w3) \rightarrow (Z1=a) \vee (Z1=c)$
$P(R1,w4) \rightarrow (R1=c) \vee (R1=d)$

//группа условий, показывающих, что исполнителям не могут назначаться одинаковые работы

$P(a,X) \& P(b,Y) \rightarrow (X<>Y)$
$P(a,X) \& P(c,Y) \rightarrow (X<>Y)$
$P(a,X) \& P(d,Y) \rightarrow (X<>Y)$
$P(b,X) \& P(c,Y) \rightarrow (X<>Y)$
$P(b,X) \& P(d,Y) \rightarrow (X<>Y)$
$P(c,X) \& P(d,Y) \rightarrow (X<>Y)$

Для этой системы знаний нужно доказать следующую формулу

$$\exists X \exists Y \exists Z \exists R \, (P(a,X) \& P(b,Y) \& P(c,Z) \& P(d,R)).$$

Продукционная модель знаний

Продукционная модель представляет собой вариант логической

модели, записанной с помощью продукционных формул (правил) вида

$$F \rightarrow G, \qquad (2.2)$$

где F,G – произвольные, вообще говоря логические формулы. Отмечается, что продукции “более близки” по форме “естественным” умозаключениям. На практике это чрезмерно общее условие можно ограничить, рассматривая формулы так называемого секвенционального вида

$$f_1 \& f_2 \& \ldots \& f_z \rightarrow g_1 \vee g_2 \vee \ldots \vee g_t, \qquad (2.3)$$

где все используемые в представлении (2.3) формулы суть атомарные. Формулу (2.3) нетрудно привести к форме дизъюнкта:

$$\neg f_1 \vee \neg f_2 \& \ldots \vee \neg f_z \vee g_1 \vee g_2 \vee \ldots \vee g_t, \qquad (2.4)$$

так что никакой особенной выгоды представление (2.4) с этой точки зрения не дает.

Система логического программирования Пролог использует еще более упрощенные секвенциальные формулы (так называемую нотацию Хорна):

$$f_1 \& f_2 \& \ldots \& f_z \rightarrow g_1 \qquad (2.5)$$

В (2.5) часть $f_1 \& f_2 \& \ldots \& f_z$ называется **телом** продукции, а g_1 - **головой**. Продукция без головы называется **целью** (*goal*). Каждая предикатная формула f_i называется в этом случае подцелью. Перейти от общего представления (2.4) к представлению (2.5) просто, если принять во внимание следующую эквивалентность

$$\neg f_1 \vee \neg f_2 \& \ldots \vee \neg f_z \vee g_1 \vee g_2 \vee \ldots \vee g_t \equiv \neg g_2 \& \neg g_3 \ldots \& \neg g_t \& f_1 \& \ldots \& f_z \rightarrow g_1 \quad (2.6)$$

Рассмотрим представление знаний в системе программирования

Пролог.

Пример1. Владимиру нравится все, что нравится Ире.

```
нравится(владимир,X):-
   нравится(ира,X).
```

Эта продукция соответствует формуле

нравится(ира,X) → нравится(владимир,X).

В Прологе тело продукции записывается после символов ":-".

Пример 2. Алексей мечтает о деньгах и карьере.

Это предложение определяет два факта:

```
мечтает(алексей, карьера).
мечтает(алексей, деньги).
```

Факты – это особый вид правил, которые не нужно доказывать. Такие правила, как можно видеть, представляют продукции без "головы".

При программировании на языке Пролог особую трудность вызывает использование рекурсивных продукций, в которых тело продукции содержит предикат, одновременно стоящий в голове продукции. Рассмотрим пример вычисления суммы чисел от 1 до заданного числа n.

```
sum(0, X, X).

sum(Z, S, X):-
Z1=Z-1,
S1=S+Z,
sum(Z1,S1,X).
```

Во-первых, заметим, что переменные в языке Пролог пишут с больших букв. Во- вторых, в данном примере назначение аргументов предиката sum таково: первый аргумент есть текущее число, второй аргумент есть промежуточная накапливаемая сумма, третий аргумент есть результирующая сумма. Продукция sum(0, X, X) является фактом, утверждающим, что если текущее число равно 0, то промежуточная накапливаемая сумма и результирующая

сумма совпадают. Вторая продукция как раз и дает пример рекурсии в Прологе. Эту продукцию следует читать таким образом:

Если (Z1=Z-1) и (S1=S+Z) и (1+2+3+…+Z1=S1) то
(1+2+3+…(Z-1)+Z=S).

Запись рекурсивных определений подчиняется некоторым общим правилам. Рекурсия обязана в конце концов сводится к простому факту. Тело рекурсии должно связывать значения аргументов доказываемой формулы с изменяемыми значениями аргументов той же формулы. В рассмотренном примере все эти характеристики рекурсии налицо.

Модель знаний на основе фреймов

Фреймы предложены Марвином Минским. Минский также ввел терминологию и язык фреймов, включающий понятия “фрейма”, “слота”, “терминала”, “значения по умолчанию”. Фрейм имеет следующую структуру:

{<имя-фрейма><имя слота$_1$ ><значение слота$_1$>
……
<имя слота$_n$ ><значение слота$_n$>}

В качестве примера рассмотрим фрейм <выбор скорости>:

{<выбор скорости>
<состояние дороги>:0.6
<состояние машины>:0.8
<состояние водителя>:0.5
}

Наша гипотетическая экспертная система должна определять оптимальную скорость автомобиля (в пределах дозволенной) с учетом состояния дороги, машины и водителя. В данном примере уже указаны конкретные значения, так что необходима процедура оценки скорости по конкретным значениям слотов. Такая процедура называется **демоном**. Процедура-демон запускается автоматически, когда фрейм удовлетворяет некоторому образцу.

Наряду с процедурами-демонами имеются процедуры-слуги, которые используются для установления значений слотам. Так, для определения состояния дороги можно было использовать следующий фрейм:

{<состояние дороги>
  <состояние покрытия>:0.5
  <видимость>:1.0
}

Такие же фреймы могли быть установлены для состояния машины и состояния водителя. Далее, например, для состояния покрытия можно использоваться фрейм:

{<состояние покрытия>
  <материал покрытия>:0.4
  <наличие влаги>:0.0
 <изношенность дороги>:0.1
}

Ясно, что организация экспертной оценки скорости требует создать все необходимые фреймы и разработать процедуру комплексной оценки. Хорошей основой для такой процедуры является метод Т.Саати.

Модель знаний на основе фреймов является сравнительно простой и легко реализуемой. В ее основу можно положить обычную базу данных с визуальным интерфейсом.

Семантические сети

В реальном мире любую ситуацию можно охарактеризовать следующим образом. Во-первых, указать, какие объекты участвуют в ситуации. Во-вторых, определить, в какие отношения вступают объекты. Наконец, указать свойства объектов и свойства отношений. Таким образом, можно передать знания об очень широком классе ситуаций. Рассмотрим пример. Дано предложение: “Студент Максимов сдал экзамен по химии”. В этой ситуации выделяем объекты: **Максимов** и **экзамен**. Отношения между объектами передаются с помощью глаголов. В нашем случае

отношением является глагол **сдать**. Свойством Максимова является <**студент**>. Свойством экзамена является **<по химии>**. Свойством отношения <сдать> является время и характер действия (в нашем случае – это прошедшее время). *Семантическая сеть* – это граф, вершины которого соответствуют объектам и понятиям, а дуги, связывающие вершины, определяют отношения между ними. В нашем случае имеем следующий граф (рис. 2.1).

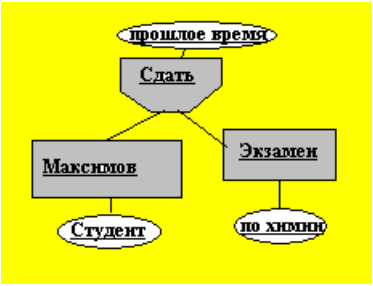

Рис. 2.1. Пример семантической сети

Объекты, отношения и свойства отображаются на семантической сети различными типами вершин. Ситуации могут образовывать целые сценарии. Например, рассмотрим второй фрагмент: “Экзамен по химии принимал профессор ZZZ”. Объединенная семантическая сеть представлена на рис. 2.2.

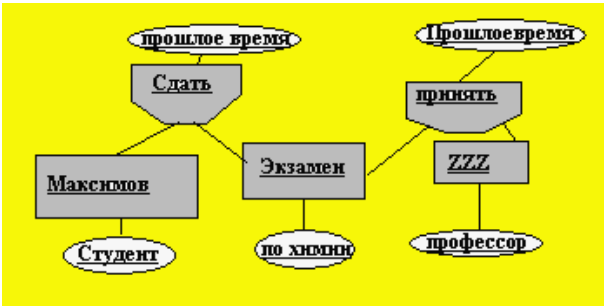

Рис. 2.2. Объединенная семантическая сеть

В современном программировании получили достаточное распространение текстовые документы XML. Язык XML есть удобное средство для описания сценариев. Так, “содержимое” рис. 2.2 “ложится” в структуру документа XML следующим образом (русские слова заменены на английские) – рис. 2.3. Документы XML обрабатываются во многих современных системах программирования, особенно в контексте Интернет-программирования.

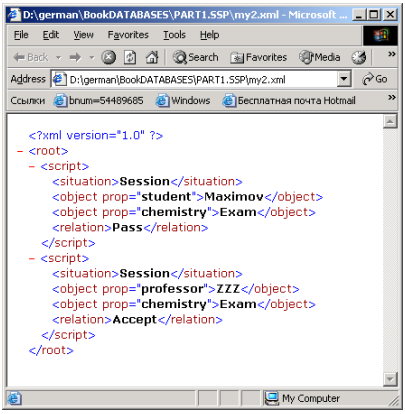

Рис. 2.3. Представление семантической сети на языке XML

Строки XML-документа представляют собой теги и их значения. Имена тегов мы связали с объектами (object), ситуациями (situation) и отношениями (relation). Свойства передаются через атрибуты тегов. Идея использования XML-документа может состоять в обработке запросов типа “Кто принимал экзамен по химии у Максимова?”. Для реализации этой идеи сам запрос можно перевести в XML-структуру:

```
<object> Maximov </object>
<object prop=”chemistry”> Exam</object>
<object> ??? </object>
<relation> Accept </relation>
```

Вопрос представляет фрагмент знаний. Значение ??? подлежит определению. Из контекста вопроса ясно, что это значение участвует в отношении Accept (Принять). Вторым объектом данного отношения является ZZZ. Именно это значение и должно быть выдано, поскольку других объектов у отношения Accept нет. Ясно, что ссылка на объект Maximov является наводящей. При обработке запроса система может выйти на связанную ситуацию, определяемую глаголом "Pass" (Сдать), в которой Maximov – действующее лицо. Таким образом, задача системы – выяснить, в какой мере обе ситуации связаны, чтобы считать Maximov существенной характеристикой запроса. Кроме того, при выполнении семантических запросов важно то обстоятельство, что отношения могут включать более двух объектов и не ясно, о каком объекте в запросе идет речь. Эту ситуацию пытаются развязать, используя падежные отношения объектов, передаваемые словами **кто**, **что**, **как**, **какой**, **чем** и пр.

ЛЕКЦИЯ 3

Логический язык. Правила построения логических формул. Понятие логического вывода

Основной задачей любой логической системы является **построение логического вывода**. Это касается как классической логики, так и неклассических логик. Для того чтобы строить выводы, нужно иметь правила вывода, а сами формулы должны быть приведены к удобному для вывода виду. В этом смысле наиболее удобно представление логических формул в виде **дизъюнктов**.

Определение. 1.[4] Дизъюнктом (в логике высказываний) называется дизъюнкция литер (булевских переменных) с отрицанием или без, например, $x \vee y \vee \neg z$. 2. Дизъюнктом в логике предикатов называется дизъюнкция литералов (атомарных формул), взятых с отрицанием или без него, например, $P(a,z) \vee \neg Q(z,f(x))$.

Определение. Конъюнктивной нормальной формой (КНФ) называется конъюнкция дизъюнктов.

Определение. Выполняющей интерпретацией I для заданной

КНФ называется множество значений переменных этой КНФ, при которых она истинна. Например, КНФ $(x_1 \vee \neg x_2)(\neg x_1 \vee x_2)(\neg x_2 \vee \neg x_3)(x_1 \vee x_3)$ истинная в интерпретации $I=<x_1=1,x_2=1,x_3=0>$ либо в интерпретации $<x_1=0, x_2=0, x_3=1>$.

Определение. Задача ВЫПОЛНИМОСТЬ (коротко – ВЫП) формулируется так: для данной КНФ установить, имеется ли для нее хотя бы одна выполняющая интерпретация.

Определение. Правилом вывода R называется такое соотношение между формулами $A_1, A_2, \ldots, A_n$ и B, которое устанавливает истинность формулы B всякий раз, когда выполняется заданное соотношение.

Определение. Формула B выводима из формул $A_1, A_2, \ldots, A_n$, если имеется конечная последовательность формул П, начинающаяся с любой из формул A_i, такая, что каждая очередная формула этой последовательности либо выводима по некоторому правилу вывода из предшествующих членов (или их части), либо совпадает с какой-то из формул A_i.

Теорема 3.1 (о полноте в узком смысле логики первого порядка и логики высказываний). Всякая тождественно истинная формула выводима.

Задача логического вывода в логике высказываний может быть эффективно сведена к ВЫП. В самом деле, пусть даны дизъюнкты $D_1, D_2, \ldots, D_z$. Спрашивается, выводим ли из них дизъюнкт R? (т.е. требуется установить тождественную истинность формулы $D_1 \& D_2 \& \ldots \& D_z \rightarrow R$. Умножим эту последнюю формулу справа и слева на $\neg R$. Получим:

$$D_1 \& D_2 \& \ldots \& D_z \& \neg R \rightarrow \text{False} \qquad (3.1)$$

Следовательно, если удастся показать выполнимость системы дизъюнктов $D_1 \& D_2 \& \ldots \& D_z \& \neg R$, то получим опровержение исходной формулы $D_1 \& D_2 \& \ldots \& D_z \rightarrow R$. Если докажем, что система не выполнима, то получим доказательство исходной формулы. Таким образом, задача логического вывода сводится к задаче ВЫП.

Приведем пример.
Пусть даны формулы

$$f_1 = a \to bc\bar{y},$$
$$f_2 = b \to \bar{e}f,$$
$$f_3 = c \vee \bar{e} \to x \vee y.$$

Покажем, что имеет место выводимость: $a \cdot f_1 \cdot f_2 \cdot f_3 \to x$.

Умножим правую и левую части на $\bar{x}$:

$$a \cdot (a \to bc\bar{y}) \cdot (b \to \bar{e}f) \cdot (c \vee \bar{e} \to x \vee y) \cdot \bar{x} \to \square.$$

Символ □ обозначает ложь ($x \cdot \bar{x} = \square$). Теперь нам надо в левой части раскрыть скобки. При этом может получиться одна из следующих ситуаций:

$\square \to \square$ – выводимость имеет место;
$S \to \square$ – выводимость не имеет места, если $S \neq \square$.

При раскрытии скобок используем известные формулы:

$$\begin{aligned} &a \to b \equiv \bar{a} \vee b, \\ &\overline{a \cdot b} = \bar{a} \vee \bar{b}, \\ &\overline{a \vee b} = \bar{a} \cdot \bar{b} \end{aligned} \qquad (3.2)$$

Две последние формулы называются формулами де Моргана. Итак, имеем

$$a \cdot (a \to bc\bar{y}) \cdot (b \to \bar{e}f)(c \vee \bar{e} \to x \vee y) \cdot \bar{x} \to \square.$$

Преобразованием получаем:

$$a \cdot (\bar{a} \vee bc\bar{y}) \cdot (\bar{b} \vee \bar{e}f)(\overline{c \vee \bar{e}} \vee x \vee y) \cdot \bar{x} = a \cdot (\bar{a} \vee bc\bar{y}) \cdot (\bar{b} \vee \bar{e}f)(\bar{c} \cdot e\bar{x} \vee y\bar{x}) =$$
$$= abc\bar{y} \cdot (\bar{b} \vee \bar{e}f)(\bar{c} \cdot e\bar{x} \vee y\bar{x}) = abc\bar{y}\bar{e}f \cdot (\bar{c} \cdot e\bar{x} \vee y\bar{x}) =$$
$$= \square .$$

Таким образом, получили ситуацию типа □ → □, т.е. выводимость имеет место.

С другой стороны, отношение $a \cdot f_1 \cdot f_2 \cdot f_3 \to y$ не выводимо (убедитесь самостоятельно).

При приведении формул к виду дизъюнктов используют, кроме указанных выше (3.2) следующие формулы:

$$\begin{aligned} &a \vee a \cdot b = a, \\ &a \vee \bar{a} b = a \vee b \\ &a \cdot (a \vee S) = a, \\ &a \cdot (\bar{a} R \vee S) = aS \\ &(a \cdot c \vee S) = (S \vee c) \cdot (a \vee S) \\ &a \cdot b \vee a \cdot \bar{b} = a. \end{aligned} \tag{3.3}$$

Пример. Привести к виду дизъюнктов следующую формулу:
$\overline{(c \vee a \cdot b)} \vee b \cdot (\bar{c} \vee \bar{b} e)$

Имеем:
$\overline{(c \vee a \cdot b)} \vee b \cdot (\bar{c} \vee \bar{b} e) = \bar{c} \cdot (\bar{a} \vee \bar{b}) \vee b \cdot \bar{c} = \bar{c}\bar{a} \vee \bar{c}\bar{b} \vee \bar{c} b = \bar{c}\bar{a} \vee \bar{c} = \bar{c}.$

Пояснение. $\overline{(c \vee a \cdot b)} = \bar{c} \cdot \overline{a \cdot b} = \bar{c} \cdot (\bar{a} \vee \bar{b})$

Сложнее обстоит дело с приведением к виду дизъюнктов в логике предикатов. Такое приведение выполняется в три этапа. **На первом этапе** все кванторы выносят в начало формулы.

Пример.
$\forall x \exists y (P(x, y) \to \exists z Q(x, z))$.

Вынесение кванторов в начало формулы дает такой результат:
$\forall x \exists y (P(x, y) \to \exists z Q(x, z)) = \forall x \exists y \exists z (P(x, y) \to Q(x, z))$.

Как видим, здесь выполняется прямолинейный перенос кванторов. Имеется, все же одна зацепка. Изменим формулу следующим образом:

$$\forall x \exists y (P(x, y) \to \exists y Q(x, y)).$$

Мы видим, что имеется два квантора существования с одной и той же связываемой переменной y. В таких ситуациях требуется переход к новым обозначениям, от которых формула не теряет своей тождественности. В нашем случае следует поступить так:

$$\forall x \exists y (P(x, y) \to \exists y Q(x, y)) = \forall x \exists y (P(x, y) \to \exists t Q(x, t)) =$$
$$= \forall x \exists y \exists t (P(x, y) \to Q(x, t))$$

Никаких недоразумений при этом не возникает.

На втором этапе отбрасывают кванторы. Здесь имеется специфика только в отношении кванторов существования. Смотрят, какие кванторы всеобщности предшествуют кванторам существования. Так, в формуле

$$\forall x \exists y \exists t (P(x, y) \to Q(x, t))$$

квантору $\exists y$ предшествует квантор всеобщности $\forall x$. Аналогично, квантору существования $\exists t$ предшествует квантор всеобщности $\forall x$. Берут самый внутренний квантор существования, и заменяют переменную t на произвольную функцию от предшествующих переменных в кванторах всеобщности, а сам квантор существования отбрасывают, например:

$$\forall x \exists y (P(x, y) \to Q(x, f(x))).$$

То же самое делают и с квантором существования $\exists y$, но используют уже другую функцию:

$$\forall x (P(x, h(x)) \to Q(x, f(x))).$$

Приведенная процедура избавления от кванторов существования называется *сколемизацией* (по фамилии норвежца Skolem). Теперь, когда кванторов существования не осталось, кванторы всеобщности просто отбрасывают:

$$P(x,h(x)) \to Q(x,\, f\,(x))\,.$$

Следующий пример применения сколемизации даем без комментариев.

$$\forall x\exists y\forall z\exists t(P(x,y) \to Q(x,t,z)) = \forall x\exists y\forall z(P(x,y) \to Q(x,\, f\,(x,z),z)) =$$
$$= \forall x\forall z(P(x,h(x)) \to Q(x,\, f\,(x,z),z)) = P(x,h(x)) \to Q(x,\, f\,(x,z),z).$$

Наконец, третья фаза состоит в получении собственно дизъюнктов на основании формул (3.2, 3.3). В нашем случае получаем единственный дизъюнкт:

$$P(x,h(x)) \to Q(x,\, f\,(x,z),z) = \overline{P}(x,h(x)) \vee Q(x,\, f\,(x,z),z).$$

Следует также указать, как строить отрицания от кванторов. Вот эти правила:

$$\begin{aligned}
&\overline{\forall} xP(x) = \exists x\overline{P}(x);\\
&\overline{\forall} x\exists y\forall zP(x,y,z) = \exists x\overline{\exists} y\forall zP(x,y,z) = \\
&\exists x\forall y\overline{\forall} zP(x,y,z) = \exists x\forall y\exists z\overline{P}(x,y,z) = \overline{P}(c,y,\, f\,(y)).
\end{aligned} \tag{3.4}$$

Здесь внимательно рассмотрите второй случай. Отрицание последовательно «распространяется» по формуле. При этом очередной квантор всеобщности заменяется на квантор существования и наоборот. В конце концов, отрицание «добирается» до самой формулы. После этого мы проводим сколемизацию. И также обращаем внимание на то, что первому квантору существования $\exists x$ не предшествует ни один квантор всеобщности. В этом случае переменная x заменяется на произвольную константу c.

В заключение этой лекции отметим в качестве универсального правила вывода в логике правило modus ponens (с латыни переводится как «правило исключения»). Пример этого правила:

Если A
и
из A следует B,
То B.

Пример.
Если (металл(x), то проводит ток(x))
металл(свинец)
Вывод: проводит ток(свинец).

В формальном виде это пишут так:

$$metal(plumbum)$$
$$\forall x\, metal(x) \rightarrow conductElectricity(x)$$
$$\_\_\_\_\_$$
$$conductElectricity(plumbum)$$

Весьма сильным правилом вывода в логике является правило приведения к абсурду (reduction ad absurdum). Формальная запись этого правила такова:

$$\alpha \rightarrow \beta$$
$$\overline{\beta}$$
$$\_\_\_\_\_$$
$$\overline{\alpha}$$

Пример.

Если стараться, добьешься результата.
Результата не добился.
Вывод: не старался.

ЛЕКЦИЯ 4

Язык Пролог как продукционная система знаний. Синтаксис языка Пролог. Примеры программ

Программа на языке Пролог состоит из секций: domains, predicates, clauses, goal. В секции domains содержатся объявления пользовательских типов данных. В секции predicates содержатся объявления предикатов. В секции clauses помещаются правила и, наконец, в секции goal записывается цель программы.

Предикаты (предикатные формулы) являются основой концепции Пролога. Выполнение программы на Прологе заключается в доказательстве целевого предиката. Этот предикат является обычно правилом. Чтобы доказать правило (любое, а не только цель), надо доказать все предикаты, составляющие его тело. Для этого происходит согласование предиката с другим одноименным предикатом, т.е. сопоставление (унификация) соответствующих аргументов этих предикатов. Согласование выполняется успешно, если успешна унификация всех аргументов предикатов. При унификации аргументов возможны следующие случаи [2]:

- сопоставление двух констант заканчивается успешно, если они равны, и неудачно – в противном случае;
- сопоставление константы и переменной (свободной, т.е. не имеющей значения) – заканчивается успешно, и переменная получает значение константы (становится связанной);
- сопоставление двух связанных переменных заканчивается успешно, если значения этих переменных равны, и неудачно, если не равны;
- сопоставление двух переменных, одна из которых связана, а другая свободна заканчивается успешно, и свободная переменная принимает значение связанной;
- сопоставление двух свободных переменных заканчивается успешно; Если впоследствии одна из переменных получает значение, то и другой переменной присваивается то же значение.

Если согласование предикатов закончилось неудачно, то делает-

ся попытка согласования данного предиката с другим одноименным клозом. Если таких клозов нет, то происходит возврат (бэктрекинг) к ближайшей “развилке”.

Рассмотрим алгебраический пример. Пусть некто сообщает, что он задумал код (шифр) из пяти двухзначных (0 или 1) цифр. Обозначим цифры X1, X2, X3, X4, X5. При этом некто приводит следующие отношения между цифрами:

```
X1+X2+X3 >= 2,
X2+X3+X4 <= 2,
X2+X4+X5 <= 1,
X3+X4+X5 >= 1,
X1+X5 <= 1.
```

Попробуем заставить Пролог найти эти цифры.

Введем предикат

```
znach(X),
```

который устанавливает значение переменной X. Определение этого предиката таково:

```
znach(B):-B=0; B=1.
```

Предикат можно прочитать так: если B = 0 или B = 1, то значением переменной B является соответственно “0” или “1” и другого быть не может. Интересно, как осмыслить предикат без аргументов? Например, рассмотрим клоз:

```
EQUATION:-
        znach(X1),
        znach(X2),
        X1+X2=2.
```

Системе предлагается подобрать такие значения для переменных X1 и X2, чтобы их сумма равнялась 2. Решение, разумеется, единственное. Но для чего нужен предикат EQUATION? Этот пре-

дикат “возглавляет целый клоз”, а потому его надо и рассматривать как структурную единицу программы. Все клозы должны быть явным образом связаны. Для этого используется раздел программы, называемый GOAL (по-английски – цель). Чтобы понять сказанное, рассмотрим уже завершенный вариант нашей программы:

```
predicates
nondeterm       equation
nondeterm       znach(integer)

goal
equation.

clauses
        equation:-
                        znach(X1),
                        znach(X2),
                        znach(X3),
                        znach(X4),
                        znach(X5),
                        X2+X3+X4<=2,
                        X2+X4+X5<=1,
                        X3+X4+X5>=1,
                        X1+X5<=1,
write(“X1=”,X1,” X2=”,X2,” X3=”,X3,” X4=”,X4,” X5=”,X5).
znach(B):-B=0; B=1.
```

Итак, мы привели законченную программу. Программа на Прологе состоит из секций PREDICATES, CLAUSES, GOAL (и др.). В секции PREDICATES объявляются предикаты и их аргументы (если есть). Стандартными типами аргументов предикатов являются integer, real, string, char, symbol, long (соответственно – целый, вещественный, строковый – берется в двойные кавычки, символьный – берется в одиночные кавычки; символьный также записывается без кавычек малыми буквами, длинный целый – для чисел, превосходящих по абсолютной величине значение 2^{16}). В целях со-

пряжения с программами в Windows дополнительно к стандартным введены типы: byte (байтовый без знака), sbyte – (байтовый со знаком), short– короткое целое, ushort – беззнаковое целое, dword – двойное слово, ulong – беззнаковое длинное целое. В секции GOAL удобнее всего объявить один единственный предикат, представляющий всю программу как таковую. (Впрочем, это не обязательно). В секции CLAUSES приводятся клозы, определяющие предикаты. Клозы для одного и того же предиката должны быть записаны вместе.

Теперь рассмотрим важнейший механизм вывода, который неявно включался в процессе выполнения программы. Этот механизм можно назвать **ветвление-возврат**. Данное в программе определение предиката znach можно переписать “более естественно”:

```
znach(B):-B=0.
znach(B):-B=1.                    (4.1)
```

Здесь использованы два клоза вместо одного, хотя смысл действий при этом не изменился. Именно, система всегда обрабатывает клозы последовательно, в порядке записи. При первом обращении к предикату znach переменная B получает значение B=0. Если это значение не позволяет удовлетворить остальным предикатам, то выполняется повторное обращение к znach и переменная B получает значение из второго клоза, т.е. B = 1. Теперь рассмотрим целиком процесс выполнения логической программы нашего примера. Первым выполняется самый первый предикат раздела GOAL – equation. Система ищет определение этого предиката в разделе CLAUSES. Это определение имеется. Далее система выбирает первое условие из определения предиката equation – znach(X1). Система пытается доказать это условие и ищет определение предиката znach в разделе CLAUSES. Таким определением является (4.1). Система использует первое определение:

```
znach(B):-B=0.
```

Рассмотрим этот момент внимательнее. В определении (4.1)

имеется две альтернативы: для B=0 и B=1. Процесс выбора одной из альтернатив называется **ветвлением**. Всегда выбирается первая по порядку из числа неисследованных альтернатив. Итак, в момент выбора альтернативы имеется два предиката znach: первый из них – это предикат znach(X1), который система пытается доказать; и второй – znach(B):-B=0, который система берет из определения. Система сопоставляет эти два предиката и пытается согласовать их аргументы. Согласование аргументов называется **унификацией**. При унификации аргументы предикатов сопоставляются в порядке их расположения. У предиката znach только один аргумент. У доказываемого – это переменная X1, у взятого из определения – это B (тоже переменная). Таким образом, выполняется привязка аргумента X1 к аргументу B. Далее выбирается условие B = 0, доказывать которое не надо, т.к. это стандартная арифметическая операция. Переменная B получает значение B = 0. Поскольку X1 привязана к B, то и X1 получает значение X1 = 0. На этом доказательство предиката znach(X1) успешно завершено. Процесс доказательства переходит на предикат znach(X2), для которого действия выполняются по аналогии, так что X2 получает значение X2 = 0. И далее, таким же путем, X3 = 0, X4 = 0, X5 = 0. Теперь система успешно проходит проверку условий

```
X2+X3+X4 <= 2,
X2+X4+X5 <= 1,
```

но проверка условия

```
X3+X4+X5 >= 1
```

заканчивается неудачей. Теперь в действие вступает механизм **возврата**. Система возвращается в точку последнего **ветвления**. Это опять соответствует определению предиката znach, использованному при выборе значения X5 = 0. На этот раз система воспользуется определением

```
znach(B):- B=1.
```

Поэтому X5=1 на этот раз и теперь уже все условия

```
X2+X3+X4<=2,
X2+X4+X5<=1,
X3+X4+X5>=1,
X1+X5<=1
```

будут успешно пройдены.

В заключение система выполнит простой стандартный вывод на экран значений переменных, полученных на этом этапе:

```
write("X1=", X1, "X2=", X2, "X3=", X3, "X4=", X4, "X5=", X5).
```

Итак, мы рассмотрели простейшую структуру программы, ее составные части и описали процесс выполнения логической программы. При этом мы выяснили, в чем состоят логические механизмы ветвления, возврата и унификации.

Прежде всего, следует помнить, что Пролог все время «занимается» доказательством предикатов. Предикаты (предикатные формулы) являются основой концепции Пролога. Если мы пишем некую формулу, например, contain(X,Y), то вкладываем в нее определенный смысл, скажем, предложение X содержит строку Y, так что для X="ядро заряжено положительно" и Y="жено положительно" мы оцениваем формулу contain(X,Y) как истинную. Однако машине следует объяснить, что такое contain(X,Y), т.е. написать правило (или правила) для данного предиката. Эти правила помещают в разделе clauses. Для раскрытия формулы, вероятно, придется использовать новые формулы, но этот процесс должен быть, в конце концов, сведен к стандартным формулам, которые машина «понимает». При этом сведении мы используем не только собственные (пользовательские) формулы, но и известные стандартные формулы.

Например, пусть дана программа такого вида:

```
goal
```

```
    like(peter,X),write(X),readchar(_).
clauses
    like(john, carrot).
    like(peter,Z):-like(john,Z).
```

Эта программа требует найти предмет, который нравится peter. В качестве цели выбирается первый предикат секции goal – like(peter,X). Пролог начинает доказывать этот предикат. В результате такого доказательства, если оно пройдет успешно, переменная X получит значение. Так вот, значения в процессе доказательства переменные получают путем подстановки и унификации. Прежде всего, Пролог начнет рассматривать правила (clauses) для доказываемого предиката like(peter,X). Сначала он выберет первое правило like(john, carrot).

Пролог сопоставит аргументы в обоих предикатах – в предикате, который доказывается, т.е. в like(peter,X), и в предикате из правила, т.е. like(john, carrot) (рис.4.1):

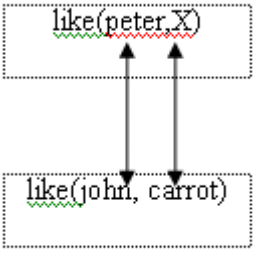

Рис. 4.1. Сопоставление аргументов в предикатах like

При сопоставлении аргументов выполняется унификация (по-русски – согласование) и присваивание значений переменным, если согласование закончилось успешно. В данном примере Пролог пытается согласовать константы peter и john и переменную X с константой carrot. С переменными дело обстоит весьма просто – они всегда согласовываются как с другими переменными, так и с константами. Поэтому Пролог примет X=carrot. С константами дело обстоит, напротив, достаточно ограничительно – просто константы

должны совпадать. Но у нас этого нет, ибо peter ≠ john. Таким образом, унификация сработала неудачно и Пролог вынужден отказаться использовать для доказательства правило like(john, carrot). Работает механизм перебора. Этот механизм заставит Пролог выбрать следующее по порядку правило для доказательства цели. Таким правилом является на этот раз

like(peter,Z):– like(john,Z).

Пролог всегда заходит в правило с его головы, т.е. выберет предикат like(peter,Z). Далее снова выполняется унификация в предикатах (рис. 4.2).

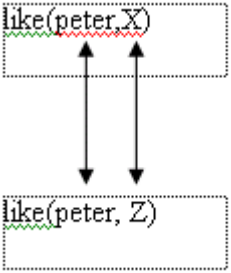

Рис. 4.2

На этот раз все благополучно и мы получим X=Z, причем Пролог с этой подстановкой зайдет в выбранное правило и начнет доказательство предикатов из его тела, т.е. будет доказывать предикат like(john,Z).

Этот новый предикат становится очередной целью. Заметим, что предикат like (…..) для своего доказательства вызывает снова предикат like(…). Такая ситуация называется **рекурсией**. Это важное понятие. Насколько оно типично для Пролога? Скажем так, не в меньшей степени, чем для любого другого языка, даже в большей ввиду широкого применения в Прологе **списков**. Но пока что мы запомним, что X=Z и очередная цель есть like(john,Z). Если эту цель удастся доказать, то найдем Z, а потому (в силу произведенной вы-

ше подстановки) и X. Для доказательства цели Пролог снова ищет набор правил с именем like. Этих правил снова два и они просматриваются в порядке расположения (рис. 4.3):

```
like(john, carrot).
like(peter,Z):-like(john,Z).
```

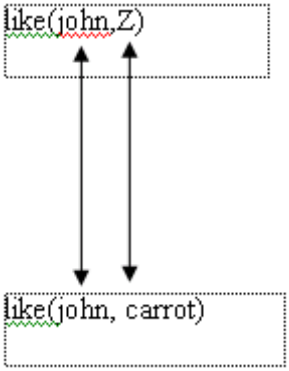

Рис. 4.3

Первые аргументы унифицировались (согласовались) и вторые тоже, т.е Z=carrot (переменная получила значение константы). Доказательство завершено, так как в данном правиле больше нечего доказывать. Пролог с успехом возвращается к предыдущей цели, т.е. к like(peter,X) и устанавливает X=Z=carrot. Тем самым цель like(peter,X) доказана. Далее в секции goal идет вывод на консоль

```
write(X),readchar(_),
```

это стандартные действия, которые не подлежат доказательству. Программа успешно завершается.

ЛЕКЦИЯ 5

Основные механизмы языка Пролог

Предикаты (предикатные формулы) являются, как говорилось выше, основой концепции Пролога. Для обработки предикатов Пролог использует следующие основные механизмы:

– подстановку и унификацию аргументов;
– перебор правил;
– отрицание и неудача;
– откат (backtracking);
– рекурсию.

Переделаем программу из предыдущей лекции следующим образом:

```
goal
   like(peter,x),write(x),readchar(_).

clauses
    like(john, carrot).
    like(john, candies):-fail.
    like(peter,Z):- not(like(john,Z)).
```

Как и ранее, программа требует найти то, что нравится peter. Однако теперь логика программы изменена – peter нравится то, что не нравится john. Это мы передали так

```
like(peter,Z):- not(like(john,Z)).
```

Отрицание передает предикат not. Однако, следует помнить, что данный предикат не может использоваться в голове, недопустимо, например,

```
not(like(peter,Z)):- (like(john,Z)).
```

Есть и второе ограничение: в момент вызова предиката с отрицанием аргументы предиката должны уже иметь значение. У нас же в программе этого нет. В like(peter,Z):- not(like(john,Z)) Z не имеет значения в момент вызова. Пролог выдаст сообщение об ошибке.

Поэтому мы переделали программу иначе:

```
predicates
nondeterm  like(symbol,symbol)
nondeterm znach(symbol)

goal
   like(peter,x),write(x),readchar(_).

clauses
    like(john, carrot).
    like(john, candies):-fail.
    like(peter,Z):- znach(Z), not(like(john,Z)).

    znach(carrot).
    znach(candies).
```

Теперь перед вызовом not(like(john,Z)) Z получит значение из вызова znach(Z). Ответ иллюстрирует рис. 5.1.

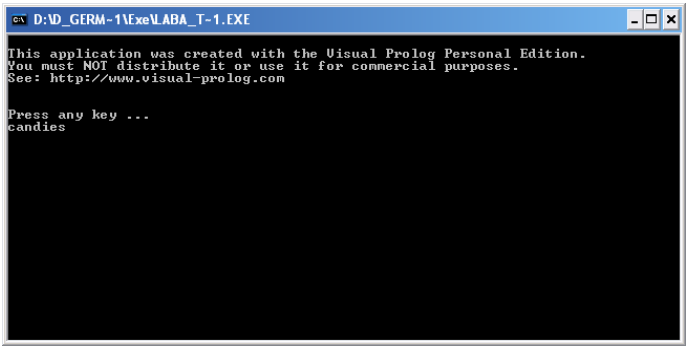

Рис. 5.1. Результат работы программы

Но как же мы передали факт, что john не любит candies? Мы сделали это так

```
like(john, candies):-fail.
```

Fail является одной из важнейших команд Пролога. Эта команда фиксирует **неудачу** в доказательстве. В нашем случае это равносильно тому, что формула like(john, candies) ложна, а потому ее отрицание истинно.

Вообще говоря, Пролог начинает поиск с возвратом, когда доказательство какого-либо предиката завершается неудачей. В определенных ситуациях бывает необходимо инициализировать поиск с возвратом, чтобы найти другое решение. Для таких целей Пролог использует специальный предикат fail-неудача.

Рассмотрим программу.

```
predicates
  nondeterm father(symbol,symbol)
  nondeterm everybody

clauses
   everybody:-
     father(X,Y),
     write(X,"is father"," ",Y), nl,
     readchar(_),fail.

father (leonard,katherine).
father (carl,nick).
father (carl,marilyn).

goal
 everybody.
```

В этом примере мы ищем отца Y. Для того, чтобы вывести **все решения**, мы в теле предиката everybody поставили команду fail.

После того, как на экран выводятся значения переменных X=leonard и Y=katherine, возникает состояние искусственной неудачи (fail). Происходит возврат к ближайшей развилке father(X,Y), т.к. этот предикат еще не сопоставлялся с двумя оставшимися фактами. При этом переменные X, Y становятся свободными. Теперь предикат father(X,Y) сопоставляется с фактом father(carl,nick). Переменные X, Y получают соответствующие значения и далее вывод осуществляется по аналогии.

Отсечение

Пролог использует отсечение для прерывания поиска с возвратом. Отсечение обозначается !. Пройдя через отсечение, уже невозможно произвести откат к предикатам, расположенным в обрабатываемом предложении перед отсечением, а также невозможно возвратиться к другим предложениям, определяющим данный предикат.

Например, есть правило:

```
r1:- a,b,
     !,
     c.
```

Если с завершится неудачей, то Пролог не сможет произвести откат (поиск с возвратом) через отсечение и найти альтернативные решения для a и b. Он также не сможет обратиться к другому предложению, определяющему предикат r1.

Пример.

```
predicates
  nondeterm buy(symbol,symbol)
  nondeterm car(symbol,symbol,integer)
  nondeterm colors(symbol,symbol)

  clauses
    buy(Model,Color):-
      car(Model,Color,Price),
      colors(Color,good),
      !,
```

```
    Price<25000,
    write(Model),
    readchar(_).

car(maserati,green,25000).
car(corvette,black,24000).
car(corvette,red,26000).

colors(red,good).
colors(black,mean).
colors(green,bad).

goal
 buy(corvette,Y).
```

В данном примере поставлена цель: найти корвет приятного цвета и подходящий по стоимости. Здесь, получив целевое утверждение buy(corvette,Y), программа отработает следующие шаги:

1. Пролог обращается к car. Выполняет проверку для первой машины. Она – maserati, следовательно, проверка завершается неудачей;

2. затем проверяет следующее предложение car и находит соответствие, связывая переменную Color со значением black;

3. переходит к следующему предикату и проверяет, имеет ли выбранная машина приятный цвет. Черный цвет не является приятным, т.о. проверка завершается неудачно;

4. выполняется поиск с возвратом к car и снова ищет Корвет;

5. находит соответствие и снова проверяет цвет. На этот раз цвет оказался нужным, затем переходим к отсечению, которое "замораживает" все переменные, ранее связанные в этом предложении;

6. переходим к сравнению Price<25000;

7. проверка завершается неудачно, и Пролог пытается совершить перебор с возвратом для того чтобы найти другую машину. Однако отсечение не даст это сделать, и наше целевое утверждение завершается неудачно.

Рекурсия

Итак, **рекурсия** - это вызов предиката из тела самого предиката. Рекурсия обычно применяется при обработке списков, при вычислениях (например, вычислениях сумм, факториала) и в ряде других случаев.

Приведем типичный пример программы с использованием рекурсии: вычисление факториала.

```
predicates
  nondeterm fact(long,long,long)

  clauses
     fact(1,N,M):-M=N.
     fact(Z,N,M):-
       N1=Z*N,
       Z1=Z-1,
       fact(Z1,N1,M).

goal
 fact(5,1,M),write("factorial=",M),readchar(_).
```

Здесь N – промежуточное значение; M – окончательное значение; Z – исходное число.

Пример факториала с двумя аргументами.

```
predicates
nondeterm fact(long,long)

clauses
 fact(1,1).
 fact(N,M):-
    N1=N-1,
    fact(N1,M1),
    M=M1*N.
goal
 fact(5,M),write("factorial=",M),readchar(_).
```

ЛЕКЦИЯ 6

Сложные структуры данных в Прологе: функторы и списки

Список представляет собой сложную структуру данных в Прологе [2].

Как ранее отмечалось, сложные типы данных (например, списки) необходимо объявлять в секции domains. **Список** - это последовательность из произвольного числа элементов некоторого типа. Ограничений на длину списка нет. Объявление списка осуществляется следующим образом:

```
domains
списочный_тип=тип*,
```

где “тип” – тип элементов списка это может быть как стандартный тип, так и нестандартный, т.е. заданный пользователем и объявленный в разделе domains ранее.

Ниже приведен пример объявления предиката, аргументом которого является список целых чисел:

```
domains
   list=integer*
 predicates
   nondeterm numbers(list)
```

Кроме того, аргументами списка могут быть не только целые числа и строки, но также и сами списки. Например,

```
domains
list=integer*
list2=string*
list3=list*

L=[1,2,3,4]
L=[“vova”,”petya”]
L=[ [1,2],[4],[] ]
```

Ввод списков с клавиатуры выполняется стандартным предикатом readterm (<тип>,<имя_переменной>), где <тип> – списочный тип, который предварительно должен быть объявлен в разделе domains. Например, для ввода списка list нужно использовать предикат readterm(list,L) [2].

При вводе списка на консоль необходимо набирать его точно так же, как он записывается, т.е. в квадратных скобках, через запятую и без пробелов.

Вывод списка на экран выполняется предикатом write(L), где L-список.

Для удобства обработки списков введены понятия головы и хвоста списка. Голова – это первый элемент списка, хвост – вся остальная часть списка. Т.о. хвост списка сам является списком. Для представления списка в виде головы и хвоста используется следующая запись:

[H|T], где H – голова списка (первый элемент), T – хвост списка (т.е. также список). Запись вида L=[X] обозначает список из одного элемента, причем голова этого списка – X, а хвост – []. Кроме того, пустой список нельзя разделить на голову и хвост.

Прежде чем рассматривать пример программы для обработки списков, приведем некоторые общие рекомендации:

- в операциях со списками практически всегда используется рекурсия. Рекурсия – это вызов предиката из самого себя;
- обычно предикаты для реализации работы со списками имеют несколько клозов. Первый из них относится к простейшему случаю (например, к операции с пустым списком, со списком из одного элемента, с головой списка). Последующие клозы относятся к более общим случаям и имеют следующее назначение: если список не соответствует простейшему случаю, рассмотренному в первом клозе, то его следует уменьшить на один элемент (обычно – исключить голову) и применить рекурсию к укороченному списку;
- предикаты, предназначенные для получения одного списка из другого, всегда имеют не менее двух аргументов: один – исходный список, другой – список-результат.

К числу основных операций со списками можно отнести следующие: проверка принадлежности к списку, добавление элемента в

писок, удаление элемента из списка, вывод суммы элементов списка и т.д. На основе этих операций реализуются другие, более сложные операции со списками.

Пример 1. Подсчет суммы элементов списка.
Начнем с примера, а затем дадим его объяснение.

```
goal
  sum([1,2,5,-7,4],0,X), write(X),readchar(_).

clauses
   sum([], Z, Y):– Y=Z.

   sum([H|T], W, Y):–
      W1=W+H,
      sum(T, W1, Y).
```

Списки в языке Пролог являются нестандартными типами данных, т.е. должны быть объявлены явно. Кроме того, в программе они помещаются в квадратные скобки. Наша задача - найти сумму элементов списка ([1,2,5,-7,4].

В такого рода задачах важно правильно «распределить» переменные. Так, одну переменную мы установили для результирующей суммы. В предикате sum([1,2,5,-7,4],0,X) – это переменная X. Вторая переменная (точнее сказать – аргумент) – это текущая накапливаемая сумма. Ей присвоено начальное значение 0. Наша идея проста – последовательно «отрывать» первый элемент списка и добавлять его к текущей накапливаемой сумме. Пролог позволяет представить список в виде [H | T]. Здесь вертикальная черта отделяет первый элемент списка (*голову* списка) от остальных элементов (*хвоста* списка). Таким образом, нетрудно догадаться о смысле второго правила

```
sum([H|T], W, Y):–
    W1=W+H,
    sum(T, W1, Y).
```

Голова H добавляется к текущей накапливаемой сумме W, получаем новую накапливаемую сумму W1 и рекурсивно вызываем предикат sum, но уже с новыми аргументами. Заметим при этом, что результирующая сумма не меняется и представлена одной и той же переменной Y. Кроме того, в Прологе нельзя писать

```
W=W+H,
```

т.к. запрещено изменять значение одной и той же переменной в правиле!

Должен быть ясен также и смысл первого правила

```
sum([], Z, Y):– Y=Z.
```

Если список опустеет, то результирующая сумма приравнивается к текущей накапливаемой сумме. Этот факт можно было передать и иначе

```
sum([], Z, Z).
```

Для объявления списочного типа в программе используется секция domains

```
domains
list=integer*

predicates
nondeterm sum(list,integer,integer)
```

Здесь объявлен списочный тип list. Указано, что list есть список целых чисел

```
list=integer*
```

Именно звездочка и определяет списочный тип.

Сумму элементов списка можно найти и через два аргумента:

```
domains
list=integer*

predicates
nondeterm sum(list,integer)

goal
  sum([1,2,5,-7,4],X), write(X),readchar(_).

clauses
   sum([],0).
   sum([H|T], S):-
   sum(T, S1),
   S=S1+H.
```

Пример 2. Вычисление минимального элемента списка.

```
domains
list=integer*

predicates
nondeterm min(list,integer,integer)

goal
  min([3,5,-2,18],3,X), write(X),readchar(_).

clauses
   min([],S,S).

   min([H|T],S ,X):-
   H>=S,
   min(T,S,X).

   min([H|T],S,X):-
   H<S,
   min(T,H,X).
```

Здесь мы последовательно отрываем голову от списка и сравниваем ее с первым элементом списка. Если первый элемент меньше головы, то он становится текущим, и предикат min рекурсивно вызываем с ним, если оторванная голова меньше первого элемента, то вызываем предикат min с головой в качестве текущего элемента. Эти действия повторяем до тех пор, пока список не опустеет, и тогда промежуточное значение становится окончательным.

Функторы представляют собой составные объекты данных в программах на Прологе. Они позволяют структурировать данные для удобства их представления и обработки. Функторы соответствуют структурам в языках типа С или записям в Паскале. Функторы объявляются в разделе domains следующим образом:

```
domains
тип=имя_функтора(типы аргументов)
```

Рассмотрим, например, факт:

```
owns(john,book("From Here to Eternity","James Jones")).
```

в котором утверждается, что у Джона есть книга "Отсюда в вечность", написанная Джеймсом Джонсом. Аналогично можно записать:

```
owns(john,horse(blacky)).
```

что означает: У Джона есть лошадь Блеки.

Составными объектами в этих двух примерах являются: book и horse.

Если вместо этого описать только два факта:

```
owns(john,"From Here to Eternity").
owns(john,blacky).
```

то нельзя было бы определить, является ли blacky названием книги

или кличкой лошади. Как раз для указания разницы между объектами мы и используем функторы book и horse.

Начнем с примера.

```
domains
author=string
title=string
kniga=book(author,title)
price=integer

predicates
nondeterm lib(kniga,price)

clauses
  lib(book("Ilf i Petrov","Zolotoy Telenok"),5000).
  lib(book("Ilf i Petrov","12 stulyev"),2500).
  lib(book("Duma","Koroleva Margo"),3500).

goal
  lib(book("Duma",X),Z),write(X," ",Z),readchar(_).
```

Функтором здесь является kniga=book(author, title). Типы author, title и price это просто переопределенные простые типы string, integer, т.е. не функторы. Функтор имеет имя и список аргументов. Так, именем нашего функтора является book. Аргументами являются author, title. Аргументами могут быть и другие функторы в том числе. Приведенная здесь программа отыскивает название книги (X), написанной Дюма, и ее цену (Z).

Одна из наиболее важных задач, для которых применяются функторы в программах на Прологе – объявление предикатов, которые в разных случаях могут принимать значения разных типов. В этом случае необходимо объявить набор функторов, называемых **альтернативным доменом**.

```
domains
альтернативный_домен = функтор_1(типы аргументов);
                       функтор_2(типы аргументов);
                        ……
                       функтор_n(типы аргументов)
```

Если затем указать альтернативный домен в качестве типа аргумента при объявлении предиката (в разделе predicates), то в качестве аргумента предиката можно будет указывать любой из функторов функтор_1,…, функтор_n.

Опять же приведем сначала пример.

```
domains
operator = plus; minus
expression=expression(integer,operator,integer)
```

Здесь объявлен альтернативный домен с именем operator. Этот домен включает два значения (функтора) – plus, minus. Кроме того, здесь объявлен функтор expression – который использует operator. Примером использования может быть вычисление выражений:

```
domains
operator = plus; minus
expression=expression(integer,operator,integer)

predicates
nondeterm calc(expression,integer)

clauses
  calc(expression(A1,plus,A2),Z):- Z=A1+A2.
  calc(expression(A1,minus,A2),Z):- Z=A1-A2.

goal
  calc(expression(5,plus,9),Z),write(Z),readchar(_).
```

Мы видим, что альтернативный домен задает варианты оператора – плюс (plus) или минус (minus). Результат работы программы показан на рис. 6.1.

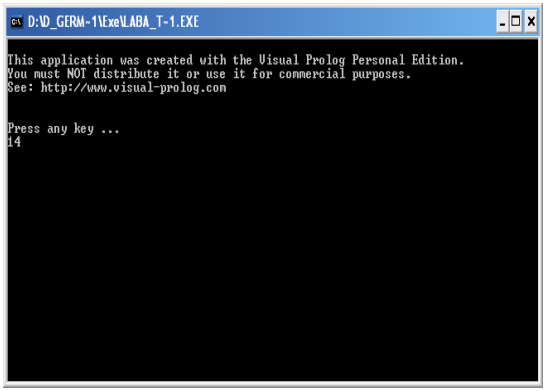

Рис. 6.1. Результат работы программы

Ввод функтора может выполняться стандартным предикатом readterm(тип, имя_переменной). Здесь "тип" – альтернативный домен, указанный в разделе domains (для рассматриваемого примера - operator). "Имя_переменной" – переменная, с которой связывается функтор. Функтор вводится в точном соответствии с его объявлением: указывается имя функтора (plus или minus) его аргументы. Аргументы функтора указываются в скобках, без пробелов, разделяются запятыми. Строковые аргументы вводятся в кавычках.

Изменим программу таким образом, чтобы ввод значений осуществлялся с клавиатуры (рис. 6.2).

```
domains
operator = plus;minus
vyrajenie=expression(integer,operator,integer)

predicates
calc(vyrajenie,integer)
```

```
clauses
  calc(expression(A1,plus,A2),Z):- Z=A1+A2.
  calc(expression(A1,minus,A2),Z):- Z=A1-A2.

goal
  write("Input number 1:\n"),readint(X),
  write("Input number 2:\n"),readint(Y),
  write("Vyraj:\n"),readterm(operator,Z1),
  calc(expression(X,Z1,Y),Z),write(Z), readchar(_).
```

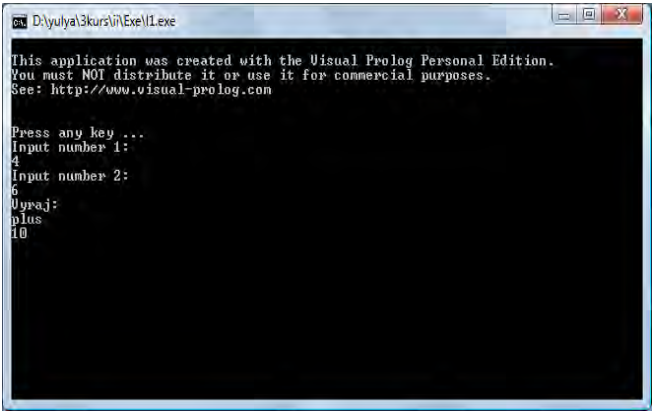

Рис.6.2. Пример ввода данных с клавиатуры

ЛЕКЦИЯ 7

Экспертные системы. Основные механизмы

Экспертные системы (ЭС) – это технологии извлечения решения задачи из базы знаний, представленных в удобном для машинной обработки виде.

Два основных механизма ЭС – база знаний и машина вывода – соответствуют двум составляющим системы решения задач – языку описания задачи и алгоритму решения задачи.

В настоящее время ЭС нашли широкое коммерческое применение, включая медицину, психологию, промышленное планирование,

теорию изобретений, САПР и пр. Задачи, которые решают ЭС, относятся к категории *интеллектуальных*.

В качестве рабочего определения экспертной системы примем следующее.

Экспертные системы - это сложные программные комплексы, аккумулирующие знания специалистов в конкретных предметных областях и тиражирующие этот эмпирический опыт для консультаций менее квалифицированных пользователей.

Обобщенная структура ЭС представлена на рис.7.1. Следует учесть, что реальные ЭС могут иметь более сложную структуру, однако блоки, изображенные на рис. 7.1, непременно присутствуют в любой ЭС, поскольку представляют собой стандарт de facto структуры современной ЭС.

В целом процесс функционирования ЭС можно представить следующим образом: пользователь, желающий получить необходимую информацию, через пользовательский интерфейс посылает запрос к ЭС; решатель, пользуясь базой знаний, генерирует и выдает пользователю подходящую рекомендацию, объясняя ход своих рассуждений при помощи подсистемы объяснений.

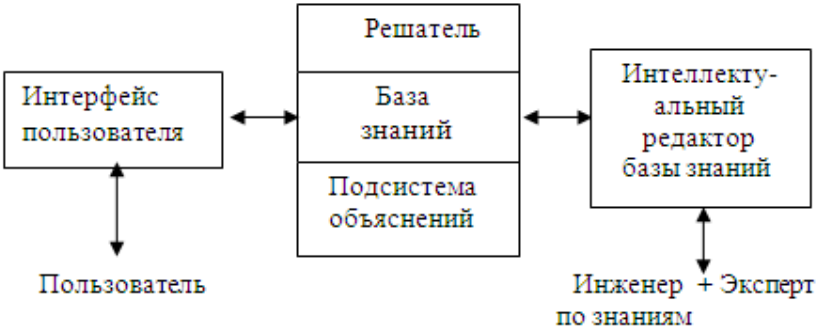

Рис. 7.1. Структура экспертной системы

Так как терминология в области разработки ЭС постоянно модифицируется, определим основные термины в рамках данной работы.

Пользователь – специалист предметной области, для которого предназначена система. Обычно его квалификация недостаточно высока, и поэтому он нуждается в помощи и поддержке своей деятельности со стороны ЭС.

Инженер по знаниям – специалист в области искусственного интеллекта, выступающий в роли промежуточного буфера между экспертом и базой знаний. Синонимы: когнитолог, инженер-интерпретатор, аналитик.

Интерфейс пользователя – комплекс программ, реализующих диалог пользователя с ЭС как на стадии ввода информации, так и при получении результатов.

База знаний (БЗ) – ядро ЭС, совокупность знаний о предметной области, записанная на машинный носитель в форме, понятной эксперту и пользователю (обычно на некотором языке, приближенном к естественному). Параллельно такому «человеческому» представлению существует БЗ во внутреннем «машинном» представлении.

Решатель – программа, моделирующая ход рассуждений эксперта на основании знаний, имеющихся в БЗ. Синонимы: дедуктивная машина, машина вывода, блок логического вывода.

Подсистема объяснений – программа, позволяющая пользователю получить ответы на вопросы: «Как была получена та или иная рекомендация?» и «Почему система приняла такое решение?». Ответ на вопрос «как» – это трассировка всего процесса получения решения с указанием использованных фрагментов БЗ, т.е. всех шагов цепи умозаключений. Ответ на вопрос «почему» – ссылка на умозаключение, непосредственно предшествовавшее полученному решению, т.е. отход на один шаг назад. Развитые подсистемы объяснений поддерживают и другие типы вопросов.

Интеллектуальный редактор БЗ – программа, представляющая инженеру по знаниям возможность создавать БЗ в диалоговом режиме. Включает в себя систему вложенных меню, подсказок («help» – режим) и других сервисных средств, облегчающих работу с базой.

Еще раз следует подчеркнуть, что представленная на рис. 7.1 структура является минимальной, что означает обязательное присутствие указанных на ней блоков. Если система объявлена разработчиками как экспертная, только наличие всех этих блоков гаран-

тирует реальное использование аппарата обработки знаний. Однако промышленные прикладные ЭС могут быть существенно сложнее и дополнительно включать базы данных, интерфейсы обмена данными с различными пакетами прикладных программ, электронными библиотеками и т.д.

Классификация систем, основанных на знаниях

Класс ЭС сегодня объединяет несколько тысяч различных программных комплексов, которые можно классифицировать по различным критериям. Полезными могут оказаться классификации, представленные на рис. 7.2.

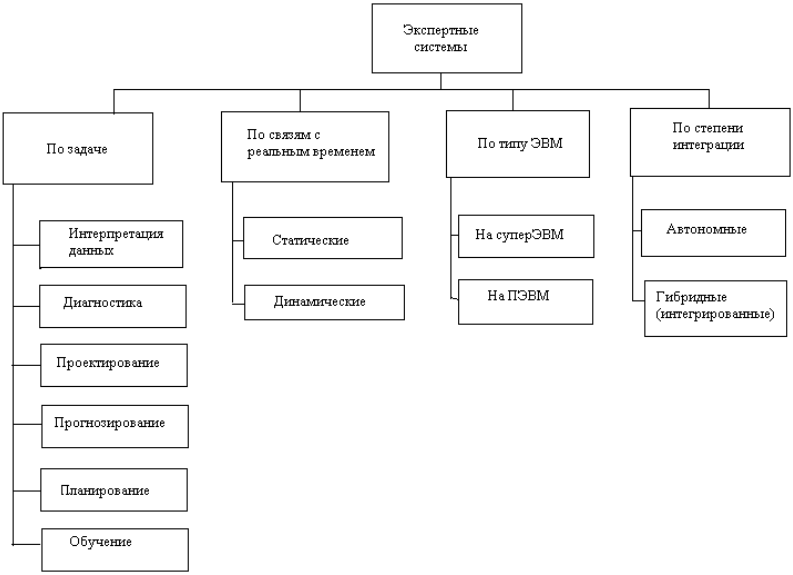

Рис. 7.2. Классификация экспертных систем

Основными механизмами любой ЭС являются База знаний и Машина Вывода. В базе знаний знания представлены в той или иной форме согласно используемой **модели**. Мы уже знаем, что такими моделями являются логические, продукционные, фреймы, се-

мантические сети (это основные). К моделям знаний можно также отнести простые таблицы данных и спецификации. Остановимся более подробно на машине вывода. Ее задача – получение ответа на вопрос задачи. Схематически это можно изобразить следующим образом (рис. 7.3).

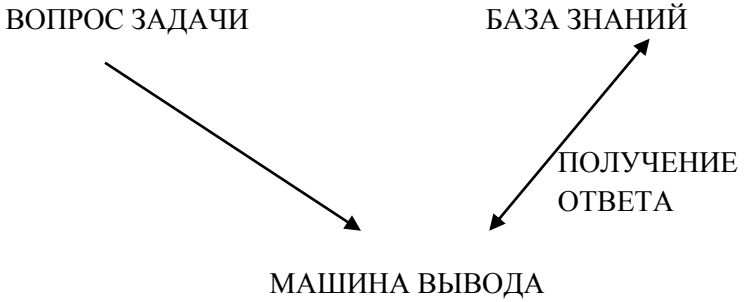

Рис. 7.3.

Реализация машины вывода «привязана» к модели знаний. Для логической и продукционной модели знаний машина вывода строит логическое доказательство (вывод). Итак, в логических моделях основной задачей является построение выводов. Для данных, представленных таблицами базы данных, строят *статистические* выводы или методы распознавания образов. Для систем неклассических логик (например, в нечеткой логике) один из подходов к реализации выводов также состоит в сведении нечеткой системы к четкой.

Построение выводов в целом эквивалентно решению задачи ВЫПОЛНИМОСТЬ.

Задача логического вывода в логике высказываний может быть эффективно сведена к ВЫП. В самом деле, пусть даны дизъюнкты D_1, D_2, …, D_z. Спрашивается, выводим ли из них дизъюнкт R? (Т.е. требуется установить тождественную истинность формулы $D_1 \& D_2 \& \ldots \& D_z \rightarrow R$). Умножим эту последнюю формулу справа и слева на $\neg R$. Получим:

$$D_1 \& D_2 \& \ldots \& D_z \& \neg R \rightarrow False.$$

Следовательно, если удастся показать выполнимость системы дизъюнктов $D_1 \& D_2 \& \ldots \& D_z \& \neg R$, то получим опровержение исходной формулы $D_1 \& D_2 \& \ldots \& D_z \rightarrow R$. Если докажем, что система не выполнима, то получим доказательство исходной формулы. Таким образом, задача логического вывода сводится к задаче ВЫП.

Для построения выводов используют правила вывода. Такими правилами вывода являются, например, modus ponens и правило построения резольвент. Решение задачи ВЫПОЛНИМОСТЬ так или иначе становится одной из центральных задач логических экспертных систем. К сожалению, данная задача является переборной (плохо решаемой). До настоящего времени нет эффективного алгоритма ее решения и полагают, что в рамках существующей математики такого решения нет вовсе. Мы рассмотрим далее один из наиболее эффективных (по отношению к другим известным алгоритмам) метод ее решения – метод (принцип) групповых резолюций (п.г.р.). Идея метода групповых резолюций состоит в одновременном порождении резольвент сразу из группы дизъюнктов, число которых в общем случае может быть равно числу переменных (вспомним, что у Робинсона в порождении резольвенты участвует только два дизъюнкта).

Метод групповых резолюций позволяет найти решение задачи о минимальном покрытии 0,1-матрицы B множеством строк. Пусть дана система дизъюнктов

$$\begin{cases} D1 = x_1 \vee \neg x_2, \\ D2 = x_2 \vee x_3, \\ D3 = \neg x_1 \vee x_2 \vee x_3, \\ D4 = \neg x_2 \vee \neg x_3. \end{cases} \qquad (7.1)$$

Для нее строим 0,1-матрицу B следующим образом. Строки матрицы соответствуют литерам x_i и их отрицаниям $\neg x_i$. Таким образом, число строк составит $2 \cdot n$, где n – число различных булевских переменных. Столбцы матрицы соответствуют дизъюнктам системы Dj, причем столбец содержит единицу в строке x_k ($\neg x_k$), если

переменная x_k входит в Dj без отрицания (с отрицанием). Кроме того, матрица дополнительно содержит n столбцов, соответствующих тавтологическим дизъюнктам $x_k \vee \neg x_k$. С учетом сказанного, матрица B для рассматриваемого примера имеет вид, изображенный на рис. 7.4.

	$D1$	$D2$	$D3$	$D4$	$D5$	$D6$	$D7$
x_1	1				1		
x_2		1	1			1	
x_3		1	1				1
$\neg x_1$			1		1		
$\neg x_2$	1			1		1	
$\neg x_3$				1			1

Рис. 7.4. 0,1-матрица B

Определение. Под *минимальным покрытием* матрицы B понимается минимальное по числу строк множество $\pi_{\min}$, такое, что для каждого столбца матрицы B найдется как минимум одна строка в $\pi_{\min}$, которая содержит в данном столбце "1" (иными словами, покрывает данный столбец).

Утверждение. Пусть дана матрица B, построенная для системы дизъюнктов с n>1 переменными. Если минимальное покрытие $\pi_{\min}$ матрицы B содержит более n строк, то данная система дизъюнктов не выполнима; в противном случае – выполнима.

Принцип групповых резолюций (п.г.р.) позволяет порождать новые – групповые резольвенты, используя любой эвристический метод для отыскания минимального или близкого к нему покрытия. Гарантируется, что рано или поздно будет порожден полностью нулевой столбец. В этом случае алгоритм прекращает работу. Наилучшее из найденных к этому моменту покрытий и является минимальным.

В качестве эвристического алгоритма можно использовать следующий. На каждой итерации p отыскиваем столбец (из числа не

вычеркнутых) с минимальным числом единиц. Обозначим этот столбец r_p и назовем его **синдромным**. Найдем невычеркнутую строку i_p, покрывающую r_p и такую, что из всех других строк, покрывающих r_p, она содержит наибольшее число единиц. Включим i_p в отыскиваемое на данной итерации p покрытие. Вычеркнем все строки, содержащие в столбце r_p "1", а также все столбцы, покрываемые строкой i_p. Итерация ведется до тех пор, пока имеется хотя бы один невычеркнутый столбец и одна невычеркнутая строка.

Так, выберем столбец $D1$ и строку $\neg x_2$, которую включим в отыскиваемое покрытие на итерации 1. Вычеркнем строки и столбцы согласно описанному правилу (рис. 7.5).

	$D2$	$D3$	$D5$	$D7$
x_1			1	
x_2	1	1		
x_3	1	1		1
$\neg x_1$		1	1	
$\neg x_3$				1

Рис. 7.5

Выполним теперь вторую итерацию. Выберем столбец $D2$ и строку x_3. Вычеркнем строки и столбцы согласно описанному правилу (рис. 7.6). Остается единственный невычеркнутый столбец, так что включим, например, строку x_1 в предполагаемое минимальное покрытие. Таким образом, итерация завершается отысканием покрытия $\pi_1 = \{\neg x_2, x_3, x_1\}$. Согласно представленному выше утверждению, данное покрытие минимально и дает выполняющую интерпретацию для исходной системы дизъюнктов.

	$D5$
x_1	1
x_2	
x_3	
$\neg x_1$	1
$\neg x_3$	

Рис. 7.6

Описанный эвристический алгоритм не всегда отыскивает минимальное покрытие и необходимо выполнять этап построения групповой резольвенты. Это делается так. Берем текущее найденное покрытие и оставляем в нем любые n+1 строку. Формируем матрицу R из *синдромных* столбцов, найденных для этих строк. Формируем столбец-резольвенту, исходя из следующего: он содержит “1” в тех и только тех строках, которые в R имеют две или более единиц; в противном случае строка содержит “0”. Этот столбец присоединяется к матрице B и итерации возобновляются снова (все вычеркнутые строки и столбцы восстанавливаем на новой итерации). Так, найденное покрытие $\pi_1 = \{\neg x_2, x_3, x_1\}$ содержит ровно $n = 3$ строки. Тем не менее, построим для него матрицу R (рис. 7.7) на синдромных столбцах $D1$, $D2$, $D5$.

	$D1$	$D2$	$D5$	Резольвента
x_1	1		1	1
x_2		1		
x_3		1		
$\neg x_1$			1	
$\neg x_2$	1			
$\neg x_3$				

Рис. 7.7

В данном случае групповая резольвента содержит единственную “1” в строке x_1. Поэтому при возобновлении итераций (если бы это было необходимо), следовало добавить данную резольвенту к матрице B.

Недостатком описанного метода является рост размеров матрицы при присоединении новых резольвент. Тем не менее, метод групповых резольвент справляется достаточно быстро с задачами, содержащими сотни логических переменных и дизъюнктов. Его эффективность падает при разреженных матрицах покрытий (с плотностью единиц порядка 0.05–0.1).

В качестве еще одного механизма рассмотрим классифицирующее дерево.

Построение классифицирующего дерева

Рассмотрим задачу поиска документов по ключевым словам. Эту задачу можно рассматривать как экспертную задачу.

Обычно при поиске документов пользователь указывает список ключевых слов для поиска. Такой поиск полностью инициируется самим пользователем. Имеется и другая возможность. Система может путем наводящего опросника выяснить, какой документ требуется пользователю. Преимуществом такого способа является более точная «наводка».

Рассмотрим следующий список документов с указанными для них множествами ключевых слов (рис. 7.8).

Документ	Ключевые слова
Д1	знание, экспертная система, решение задач
Д2	знание, диагностика, алгоритм, решение задач, классификация
Д3	база данных, язык программирования, запрос
Д4	алгоритм, сложность,решение задач
Д5	алгоритм, классификация

Рис. 7.8. Пример списка документов

Чтобы построить минимальное по размеру классифицирующее дерево, составим матрицу различий, столбцами которой являются

пары документов, а строками – ключевые слова. В строке α и столбце β пишем “1”, если и только если это ключевое слово различает данные документы, т.е. в одном из них оно присутствует в качестве ключевого слова, а в другом нет. Например, ключевое слово «алгоритм» различает документы Д1 и Д4 (у Д1 нет, а у Д4 есть ключевое слово «алгоритм»).

Мы приняли для слов сокращения по первым буквам, например, знание – зн, экспертная система – эс и т.д.

Матрица различий имеет следующий вид (рис. 7.9).

	Д1, Д2	Д1, Д3	Д1, Д4	Д1, Д5	Д2, Д3	Д2, Д4	Д2, Д5	Д3, Д4	Д3, Д5	Д4, Д5
зн		1	1	1	1	1	1			
эс	1	1	1	1						
рз		1		1	1		1	1		1
диа	1				1	1	1			
алг	1		1	1	1			1	1	
бд		1			1			1	1	
яп		1			1			1	1	
запр		1			1			1	1	
слож			1			1		1		1
класс	1			1	1	1			1	1

Рис. 7.9. Матрица различий

Определим покрытие матрицы как множество строк, таких, что для любого столбца хотя бы одна из строк покрытия содержит в нем “1”. Покрытие минимально, если оно содержит минимальное количество строк. Для отыскания минимального покрытия как раз и можно использовать описанный выше метод групповых резолюций. Одно из минимальных покрытий нашей матрицы имеет вид: π = {зн, алг, класс}. Именно с помощью этих трех слов (знание, алгоритм, классификация) можно однозначно определить документ. Пример соответствующего опросника:

– говорится ли в документе про «алгоритм»
(Ответ – Нет)
– говорится ли в документе про «знание»
(Ответ – ДА)
Система сообщает – Это Д1.

Этот опрос можно представить в виде дерева (рис. 7.10).

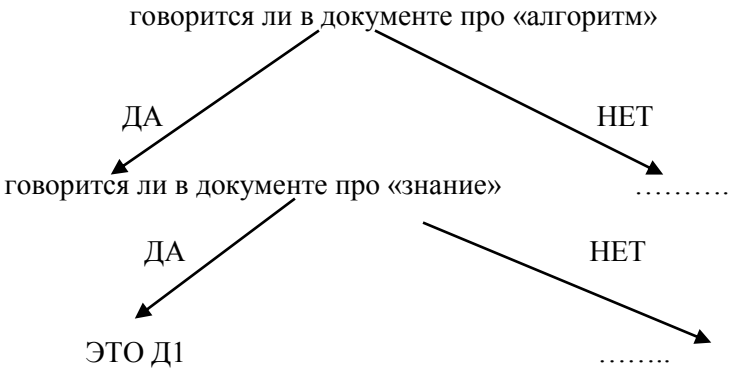

Рис. 7.10. Пример опросника

Разумеется, при большом (сотни и тысячи) числе документов и числе ключевых слов порядка нескольких сотен построение классифицирующего дерева может быть сопряжено со значительными техническими трудностями, так что речь следует вести не о дереве минимального размера, а о близком к минимальному по размерам дереву.

ЛЕКЦИЯ 8

Основы нечеткой логики

Основы нечеткой логики предложены Л. Заде в 60-х годах. Рассмотрим, например, следующую логическую формулу

$$\bar{x}_1 \vee x_2 \qquad (8.1)$$

Эта формула при одних значениях переменных может быть истинной, а при других – ложной. Если о значениях переменных ничего не известно, то в предположении их равновероятности, можно оценить вероятность формулы быть истинной, как 0.75 (три случая из четырех). С другой стороны, если переменные не равновероятны, то вероятность этой же формулы быть истинной уже может быть не 0.75, а например, 0.9 или 0.6. На практике мы как раз не всегда знаем вероятности переменных быть истинными, а используем вместо этого *субъективные* оценки. Такие субъективные оценки выражаются словами типа «достаточно правдоподобно», «трудно отдать предпочтение чему-либо», «мало вероятно» и т.п. Подобные оценки называются **лингвистическими**. Они не являются численными. Однако их следует перевести к числовому виду. Для этого используют **нечеткую функцию меры**. Нечеткую функцию меры обозначают, как правило, символом μ. Она сопоставляет лингвистическим значениям числовые. Такое сопоставление субъективно. Оно не основано на опытных статистических данных, а характеризует представление эксперта. Между прочим, такие субъективные оценки называются также **субъективными вероятностями**. Если, например, мы оцениваем переменную x_1 как истинную с вероятностью 0.8, а переменную x_2 как истинную с вероятностью 0.4, то какова субъективная вероятность формулы (8.1)? Ведь при этом мы уже не опираемся на какой бы то ни было статистический материал. На помощь нам может прийти диаграмма Венна для решения этой проблемы. Условно можно разделить предметы на четыре части: *A*, *B*, *C*, *D*. Часть *A* составят предметы, у которых нет ни свойства $\bar{x}_1$ ни свойства x_2. Для них формула (8.1) ложна. Часть *B* составят предметы, у которых нет свойства $\bar{x}_1$ но есть свойство x_2. Часть *C* составят предметы, у которых есть свойство $\bar{x}_1$ но нет свойства x_2. Наконец, часть *D* составят предметы, обладающие обоими свойствами. Обозначим через m_a, m_b, m_c, m_d – числа предметов в каждой части соответственно. Тогда свойством (8.1) обладает число

предметов, равное $m_b + m_c - m_d$. Это простое соображение дает нам первую отправную позицию для исчисления нечетких оценок истинности формул типа (8.1):

$$\mu(\bar{x}_1 \vee x_2) = \mu(\bar{x}_1) + \mu(x_2) - \mu(\bar{x}_1 \,\&\, x_2). \qquad (8.2)$$

Считая переменные x_1, x_2 независимыми, получаем:

$$\mu(\bar{x}_1 \vee x_2) = \mu(\bar{x}_1) + \mu(x_2) - \mu(\bar{x}_1) \cdot \mu(x_2) \qquad (8.3)$$

Формула (8.3) называется формулой Шортлифа [5]. Она легко обобщается на случай трех и более переменных. Следующая формула также постулируется без доказательств:

$$\mu(\bar{x}_1) = 1 - \mu(x_1) \qquad (8.4)$$

Заметим, что Л.Заде использовал не формулу (8.3), а формулу

$$\mu(\bar{x}_1 \vee x_2) = \max\{\mu(\bar{x}_1); \mu(x_2)\}. \qquad (8.5)$$

Насколько (8.3) отличается от (8.5) можно судить из следующей табл. 8.1.

Таблица 8.1

$\bar{x}_1$	x_2	$\mu(\bar{x}_1 \vee x_2)$(8.3)	$\mu(\bar{x}_1 \vee x_2)$(8.5)	расхождение
0	0	0	0	0
0	0.1	0.1	0.1	0
0	0.3	0.3	0.3	0
0.1	0.6	0.64	0.6	0.04
0.2	0.8	0.84	0.8	0.04
0.5	1	1	1	0
0.6	0.6	0.84	0.6	0.24
0.8	0.8	0.96	0.8	0.16
0.9	0.9	0.99	0.9	0.09
1	1	1	1	0

Итак, наиболее значительное расхождение имеет место, когда меры истинности переменных близки друг к другу и группируются в области значений от 0.5 и выше.

Наконец, Заде использовал формулу

$$\mu(\bar{x}_1 \,\&\, x_2) = \min\{\mu(\bar{x}_1); \mu(x_2)\}, \tag{8.6}$$

хотя вероятностным аналогом этой формулы является

$$\mu(\bar{x}_1 \,\&\, x_2) = \mu(\bar{x}_1) \cdot \mu(x_2 \mid \bar{x}_1). \tag{8.7}$$

Опять же, если считать переменные независимыми, то получим

$$\mu(\bar{x}_1 \,\&\, x_2) = \mu(\bar{x}_1) \cdot \mu(x_2) \tag{8.8}$$

Собственно, (8.3), (8.4) и (8.8) дают нам возможность строить машину нечеткого логического вывода на сравнительно ясной математической платформе. Это можно показать на примере. Пусть дана система нечетких логических дизъюнктов

$$\begin{aligned} D_1 &= x_1 \vee \bar{x}_2 \quad (0.6), \\ D_2 &= \bar{x}_1 \vee \bar{x}_2 \quad (0.9) \end{aligned} \tag{8.9}$$

Требуется найти нечеткое решение этой системы. Запишем систему уравнений:

$$0.6 = \mu(x_1 \vee \bar{x}_2) = \mu(x_1) + \mu(\bar{x}_2) - \mu(x_1) \cdot \mu(\bar{x}_2)$$
$$0.9 = \mu(\bar{x}_1 \vee \bar{x}_2) = \mu(\bar{x}_1) + \mu(\bar{x}_2) - \mu(\bar{x}_1) \cdot \mu(\bar{x}_2)$$

Воспользовавшись (8.4), получим в итоге [6]

$$0.6 = \mu(x_1) + 1 - \mu(x_2) - \mu(x_1) \cdot (1 - \mu(x_2))$$
$$0.9 = 1 - \mu(x_1) + 1 - \mu(x_2) - (1 - \mu(x_1)) \cdot (1 - \mu(x_2))$$

Здесь ровно две переменные и два уравнения (уравнения нелинейные). Кроме того, следует добавить еще ограничения на значения переменных:

$$0 \le \mu(\bar{x}_1) \le 1; \quad 0 \le \mu(x_2) \le 1.$$

Такую систему легко решить в Excel с помощью пакета Поиск Решения.

a	b
0,2	0,499999
0,399999	
1,1	
0,2	
0,2	
0,499999	
0,499999	

Мы получили ответ: $\mu(x_1) = 0.2; \quad \mu(x_2) = 0.5$

Если использовать соотношение Заде (8.5), то примем во внимание, что $\max(x, y) = 0.5 \cdot [abs(x + y) + abs(x - y)]$.

Работа с абсолютными значениями (*abs*) является крайне неудобной по техническим соображениям.

Например, $abs(x + y) = \sqrt{(x + y)^2}$. Поэтому нотацию Заде следует признать плохо приспособленной к вычислениям. Мы далее не будем ее рассматривать в практическом плане.

Введем в систему еще один дизъюнкт:

$$\begin{aligned} D_1 &= x_1 \vee \bar{x}_2 \quad (0.6), \\ D_2 &= \bar{x}_1 \vee \bar{x}_2 \quad (0.9), \\ D_3 &= x_1 \vee x_2 \quad (0.7) \end{aligned} \qquad (8.10)$$

Теперь поиск решения не может найти решения. Поэтому мы

ищем приближенное решение, введя новые переменные, как показано ниже

$$\begin{aligned} &D_1 \leftrightarrow x_1 \vee \bar{x}_2, \\ &D_2 \leftrightarrow \bar{x}_1 \vee \bar{x}_2, \\ &D_3 \leftrightarrow x_1 \vee x_2, \\ &(D_1 - 0.6)^2 + (D_2 - 0.9)^2 + (D_3 - 0.7)^2 \to \min \end{aligned} \tag{8.11}$$

Эта система перепишется следующим образом:

$$\begin{aligned} &D_1 \to x_1 \vee \bar{x}_2, \\ &D_1 \leftarrow x_1 \vee \bar{x}_2, \\ &D_2 \to \bar{x}_1 \vee \bar{x}_2, \\ &D_2 \leftarrow \bar{x}_1 \vee \bar{x}_2, \\ &D_3 \to x_1 \vee x_2, \\ &D_3 \leftarrow x_1 \vee x_2, \\ &(D_1 - 0.6)^2 + (D_2 - 0.9)^2 + (D_3 - 0.7)^2 \to \min \\ &0 \le x_i \le 1, \\ &0 \le D_i \le 1. \end{aligned} \tag{8.12}$$

Теперь нам нужно выяснить, что такое в нечеткой логике операция следования (импликации) – $\to$. Здесь мы не выходим за рамки общепринятого подхода:

$$\alpha \to \beta \equiv \mu(\alpha) \le \mu(\beta). \tag{8.13}$$

Это дает окончательно:

$$
\begin{aligned}
&\mu(D_1) = \mu(x_1) + \mu(\bar{x}_2) - \mu(x_1) \cdot \mu(\bar{x}_2),\\
&\mu(D_2) = \mu(\bar{x}_1) + \mu(\bar{x}_2) - \mu(\bar{x}_1) \cdot \mu(\bar{x}_2),\\
&\mu(D_3) = \mu(x_1) + \mu(x_2) - \mu(x_1) \cdot \mu(x_2),\\
&(\mu(D_1) - 0.6)^2 + (\mu(D_2) - 0.9)^2 + (\mu(D_3) - 0.7)^2 \to \min \qquad (8.14)\\
&0 \le \mu(x_i) \le 1,\\
&0 \le \mu(D_i) \le 1.
\end{aligned}
$$

Решим эту систему с помощью пакета Поиск решения и найдем:

D1	D2	D3	mx1	mx2
0,583307	0,859066	0,669432	0,252739	0,557627

-4,9E-08	
4,92E-08	0,002889
4,92E-08	
0,252739	
0,252739	
0,557627	
0,557627	

Итак,

$\mu(x_1) = 0.253; \quad \mu(x_2) = 0.557.$

Возникает вопрос, насколько найденное решение хорошо или плохо? Этот же вопрос возникает и тогда, когда число уравнений больше числа переменных или наоборот. Таким образом, общий случай решения нечетких систем уравнений (дизъюнктов) требует искать приближенное решение, наилучшим образом соответствующее заданным значениям. Степень доверия найденному решению можно оценить с помощью статистического критерия, например, χ^2. Вычисляем расчетное значение критерия χ^2 по формуле:

$$X^2 = \sum_{j=1}^{k} \frac{(n_j - E_j)^2}{E_j} \approx X^2_{k-1} \qquad (8.15)$$

Здесь n_j – фактическое значение j-го уравнения (значение уравнения можно связать с вероятностью принятия им истинного значения); E_j – заданная мера истинности j-го уравнения. Данное уравнение позволяет найти расчетное значение критерия χ^2. В нашем случае имеем

$$\chi^2 = \frac{(0.6-0.58)^2}{0.6} + \frac{(0.86-0.9)^2}{0.9} + \frac{(0.67-0.7)^2}{0.7} = 0.0038$$

Это значение следует сравнить с табличной величиной χ^2 для числа степеней свободы, равной числу переменных, поскольку только переменные определяют значения уравнений (но не наоборот). В нашем случае число степеней свободы p равно 3. Должно выполняться соотношение

расчетное значение χ^2 не выше табличного.

Для этой цели можно воспользоваться статистическими функциями Excel, именно функцией ХИ2ОБР (рис 8.1).

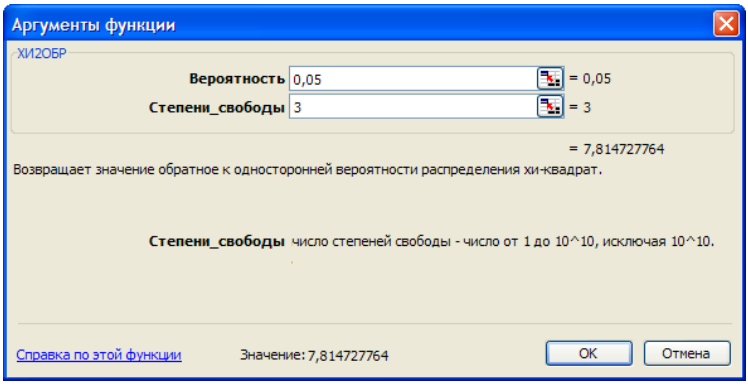

Рис 8.1. Табличное значение χ^2

Используя эту функцию, следует ввести число степеней свободы p и вероятность ошибки, например, 0.05. Это значение x=7.814 и будет соответствовать χ^2 табличному. Так, в нашем случае заключаем, что найденное нами точное решение можно рассматривать как *статистически адекватное*.

Теперь можно с рассмотренных позиций указать, как строить *нечеткий логический вывод*. Пусть имеется множество формул α_1, α_2, …, α_n, для каждой из которых известна нечеткая мера истинности: $\mu(\alpha_1)$,…, $\mu(\alpha_n)$. Спрашивается, выводима ли из этого множества формула β с нечеткой мерой истинности $\mu(\beta)$? Согласно общему правилу, выводимость трактуется так
$\alpha_1, \alpha_2, \ldots, \alpha_n \rightarrow \beta$, если и только если $\mu(\alpha_1 \,\&\, \alpha_2 \,\&\, \ldots \&\, \alpha_n) \le \mu(\beta)$ в любой интерпретации.

Так, относительно системы (8.14) спросим, выводима ли из нее формула $D_4 \leftrightarrow x_1 \quad (0.6)$? Присоединим к системе отрицание доказываемой формулы и попытаемся найти решение. Если нам удастся найти **статистически адекватное** (по критерию χ^2) решение, то выводимость места не имеет. Если решения найти не удастся, то выводимость имеет место. Итак, нам потребуется решить систему

$$\mu(D_1) = \mu(x_1) + \mu(\bar{x}_2) - \mu(x_1)\cdot\mu(\bar{x}_2),$$
$$\mu(D_2) = \mu(\bar{x}_1) + \mu(\bar{x}_2) - \mu(\bar{x}_1)\cdot\mu(\bar{x}_2),$$
$$\mu(D_3) = \mu(x_1) + \mu(x_2) - \mu(x_1)\cdot\mu(x_2),$$
$$\mu(D_4) = \mu(\bar{x}_1)$$
$$(\mu(D_1) - 0.6)^2 + (\mu(D_2) - 0.9)^2 + (\mu(D_3) - 0.7)^2 + (\mu(D_4) - 0.4)^2 \rightarrow \min$$
$$0 \le \mu(x_i) \le 1,$$
$$0 \le \mu(D_i) \le 1.$$

При этом мы добавили в систему формулу

$$\mu(D_4) = \mu(\bar{x}_1)$$

и изменили критерий

$$(\mu(D_1)-0.6)^2+(\mu(D_2)-0.9)^2+(\mu(D_3)-0.7)^2+(\mu(D_4)-0.4)^2 \to \min.$$

ЛЕКЦИЯ 9

Нечеткая математика в принятии решений

Рассмотрим достаточно простую модель принятия решений на основе следующего уравнения:

$$Y = X \otimes R_{xy}. \qquad (9.1)$$

Здесь Y – нечеткий выходной вектор, а X – нечеткий входной вектор. Нечеткий вектор – этот математический объект типа следующего:

<низкий(0.3), средний (0.6), высокий (0.1)>

Числа в скобках указывают нечеткую меру истинности данного значения в представлении эксперта или лица, принимающего решения. Например, <низкий (0.3)> означает, что эксперт оценивает некое качество как низкое с оценкой 0.3. Оценки выставляются в диапазоне от 0 до 1. Наивысшая оценка 1 соответствует варианту четкой истины. Впрочем, разница между 0.99 и 1.0 нами вообще никак не воспринимается. Среднее значение 0.5 воспринимается как точка безразличия, когда нельзя ни считать наличие свойства, ни его отсутствие истинными (ложными). Любое значение нечеткой меры, большее 0.5, должно восприниматься как правдоподобно истинное, а меньшее 0.5 – как правдоподобно ложное.

В выражении (9.1) R_{xy} называется матрицей нечеткого отношения «вход-выход». Для удобства восприятия рассмотрим такой пример. Пусть входной вектор X характеризует спрос на товар, например

$X =$ <низкий(0.3), средний (0.6), высокий (0.1)> (9.2)

Выходной вектор Y характеризует цену, которая условно может принимать три значения: *низкая*, *средняя* и *высокая*. Зададим матрицу нечеткого отношения «спрос-цена» (рис. 9.1) следующим образом (матрица составляется экспертом, т.е. является субъективной характеристикой).

$R_{xy} =$

X/*Y*	Цена низкая	Цена средняя	Цена высокая
Спрос низкий	0.9	0.7	0.1
Спрос средний	0.6	0.9	0.6
Спрос высокий	0	0.4	0.9

Рис. 9.1. Матрица нечеткого отношения

Рассмотрим ячейку [спрос низкий, цена низкая] = 0.9. Эта оценка свидетельствует о том, что весьма разумно при низком спросе держать низкие цены (разумеется, в пределах допустимого). Впрочем, более точно проводить измерение не по шкале «спрос-цена», а по шкале «спрос-цена*объем_выпуска». Общая формула примерно такова

$$цена \cdot объем \approx \text{const} \qquad (9.3)$$

В связи с этим замечанием нашу таблицу следовало бы сделать трехмерной, но это слишком усложнило бы рассмотрение.

Итак, пусть дан входной вектор (9.3). Умножим его на таблицу (матрицу) R_{xy} по правилам нечеткого векторно-матричного умножения. Опять же в интересах простоты умножение будем выполнять традиционно, считая, что нечеткое умножение и обычное умножение совпадают. Нечеткое же сложение определим таким образом:

$$a \oplus b = \max(a, b) \qquad (9.4)$$

Умножение вектора на матрицу мы иллюстрируем таким образом:

$$<0.3,0.6,0.1>*\begin{vmatrix}0.9 & 0.7 & 0.1\\ 0.6 & 0.9 & 0.6\\ 0 & 0.4 & 0.9\end{vmatrix}=<0.3\cdot 0.9\oplus 0.6\cdot 0.6\oplus 0.1\cdot 0,\quad 0.3\cdot 0.7\oplus 0.6\cdot 0.9\oplus 0.1\cdot 0.1,$$

$$0.3\cdot 0.1\oplus 0.6\cdot 0.6\oplus 0.1\cdot 0.9>=\quad <0.36,0.54,0.36>$$

Теперь пусть цена на товар может быть установлена в приемлемом диапазоне [100, 400], где 100 – минимально приемлемая цена, а 400 – максимально возможная цена. Нами получен выше выходной вектор <низкая(0.36), средняя (0.54), высокая (0.36)>. Окончательный выбор цены произведем по формуле

$$\frac{\sum \mu_i \cdot Ц_i}{\sum \mu_i}=\frac{0.36\cdot 100+0.54\cdot 250+400\cdot 0.36}{0.36+0.54+0.36}=250 \qquad (9.5)$$

Таким образом, при указанном входном векторе и матрице нечеткого отношения <вход-выход> определено конкретное значение выходной цены на товар, равное 250.

Рассмотренный подход не является единственно возможным. Можно рассмотреть задачу принятия многокритериальных нечетких решений в следующей постановке. Пусть даны альтернативы: A_1, A_2, A_3 и матрица нечеткого отношения предпочтения на множестве альтернатив (рис. 9.2):

	A_1	A_2	A_3
A_1	0	0.7	0.5
A_2	0.1	0	0.1
A_3	0.4	0.8	0

Рис. 9.2. Матрица нечеткого отношения на множестве альтернатив

Эту матрицу следует читать таким образом. В ячейках записаны величины, определяющие степень (нечеткой) предпочтительности одной альтернативы против другой. Например, альтернатива A_1 предпочтительней A_2 с нечеткой оценкой 0.7. Наоборот,

альтернатива A_2 предпочтительнее альтернативы A_1 с нечеткой оценкой 0.1 (должно выполняться соотношение $\mu_{ij} \le 1 - \mu_{ji}$). Спрашивается, какую альтернативу выбрать?

Для решения этой задачи, опять же, можно использовать разные подходы. Один из них просто находит сумму оценок в каждой строке и выбирает альтернативу с максимальной суммой (у нас A_1 или A_3). С другой стороны, насколько, например, предпочтительность с оценкой 0.55 убедительна?

Поэтому можно предложить следующую процедуру. Если мера предпочтительности $\mu_{ij} < 0.5+\Delta$ (где Δ можно назвать порогом чувствительности), то в матрице заменяем это число на 0, в противном случае, т.е. при $\mu_{ij} \ge 0.5+\Delta$ записываем в ячейку 1. В статистике используют, например, максимальную вероятность ошибки на уровне 5%. Поэтому примем Δ=0.05 (1*0.05). Итак, пороговым значением будет 0.55. В итоге наша матрица примет такой вид (рис. 9.3):

	A_1	A_2	A_3
A_1	0	1	0
A_2	0	0	0
A_3	0	1	0

Рис. 9.3

Теперь выбираем те альтернативы, которые содержат в своих строках максимальное число 1. Наш выбор такой: A_1 , A_3. Если число выбранных альтернатив превосходит 1, то строим подматрицу на этих альтернативах, выбирая значения из ячеек исходной матрицы (рис. 9.4):

	A_1	A_3
A_1	0	0.5
A_3	0.4	0

Рис. 9.4

Теперь опять находим сумму элементов в строках и выбираем альтернативу с максимальной суммой, т.е. A_1.

Обратимся к общему случаю. Пусть имеется несколько критериев и для каждого критерия имеется соответствующая матрица нечеткого отношения предпочтения (рис. 9.5). Например:

Критерий 1:

	A_1	A_2	A_3
A_1	0	0.6	0.4
A_2	0.3	0	0.2
A_3	0.55	0.6	0

Критерий 2:

	A_1	A_2	A_3
A_1	0	0.3	0.5
A_2	0.4	0	0.7
A_3	0.4	0.2	0

Рис. 9.5. Матрицы нечеткого отношения для нескольких критериев

Находим распределение мест по критериям. Для первого критерия получаем матрицу (рис. 9.6):

	A_1	A_2	A_3	*места*
A_1	0	1	0	2
A_2	0	0	0	3
A_3	1	1	0	1

Рис. 9.6

Для второго критерия аналогичная таблица имеет такой вид (рис. 9.7):

	A_1	A_2	A_3	*места*
A_1	0	0	0	2-3
A_2	0	0	1	1
A_3	0	0	0	2-3

Рис. 9.7

Для уточнения мест в последней таблице строим подматрицу на тех альтернативах, для которых вопрос о месте не решен однозначно (рис. 9.8):

	A_1	A_3	*места*
A_1	0	0.5	2
A_3	0.4	0	3

Рис. 9.8

Итак, распределение мест по критериям таково (рис. 9.10):

	Критерий 1	Критерий 2
A_1	2 (сумма баллов 1)	2 (сумма баллов 0.8)
A_2	3 (сумма баллов 0.5)	1 (сумма баллов 1.1)
A_3	1 (сумма баллов 1.15)	3 (сумма баллов 0.6)

Рис. 9.10. Распределение мест по критериям

В качестве итогового ответа выберем альтернативу, набравшую наименьшую сумму мест, а при равенстве этого показателя – альтернативу с максимальной суммой баллов. В нашем примере в качестве ответа будет A_1.

В заключение рассмотрим, как рассчитываются нечеткие меры. Одним из таких методов является **метод интервалов**. Он предполагает, что имеется несколько экспертов, каждый из которых

ставит интервальную оценку для меры. Например, путь имеется два эксперта A и B, которые поставили такие оценки (рис. 9.11).

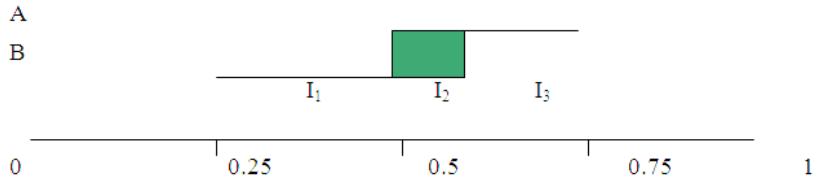

Рис. 9.11. метод интервалов

Оценка A есть интервал [0,25;0.6].
Оценка B есть интервал [0,5;0.75].
Зеленым цветом выделен общий подинтервал – [0.5; 0.6].

Разобьем длину всего интервала [0.25; 0.75] на подинтервалы:
I_1 = [0.25; 0.5]; I_2 = [0.5; 0.6]; I_3 = [0.6;0.75].
Эти подинтервалы выбраны таким образом, что на всей их длине представлена оценка одного и того же (одних и тех же) экспертов. Так, на подинтервале I_1 имеется только оценка эксперта A.
Теперь найдем середины подинтервалов I_1, I_2, I_3:
$\theta_1 = 0.375, \theta_2 = 0.55; \theta_3 = 0.675$
Пусть веса экспертов одинаковы: $\lambda_A = \lambda_B = 0.5$
Пересчитаем веса подинтервалов по формуле Шортлифа:

$$\lambda_1 = \lambda_3 = \lambda_A = \lambda_B = 0.5,$$
$$\lambda_2 = \lambda_A + \lambda_B - \lambda_A \cdot \lambda_B = 0.5 + 0.5 - 0.25 = 0.75.$$

Теперь итоговое значение нечеткой оценки произведем по формуле:

$$\mu = \frac{\sum \theta_i \cdot \lambda_i}{\sum \lambda_i} = \frac{0.375 \cdot 0.5 + 0.55 \cdot 0.75 + 0.675 \cdot 0.5}{0.5 + 0.5 + 0.75} \approx 0.56$$

ЛЕКЦИЯ 10

Нечеткий Пролог

Рассмотрим следующее предложение языка Пролог:

```
suit (peter, X): –
   big_salary (X,_)),
   good_conditions (X, _).

   big_salary(mailer, 0.4).
   big_salary(officer, 0.7).
   big_salary(bob, 0.8).
   big_salary(artist, 1.0).

   good_conditions(mailer, 0.7).
   good_conditions(officer, 0.3).
   good _conditions(bob, 0.6).
   good_conditions(artist, 0.9).
```

В нечетком Прологе вывод осуществляется таким образом, чтобы либо найти ответ с максимальной правдоподобностью, либо “обычным образом”, но учитывая, что значение 0.5 и ниже в качестве меры истинности рассматривается как ложное. Рассмотрим, как реализовать два этих подхода.

Подход по принципу 0.5 и ниже – ложь.

В этом случае наша программа должна быть переписана следующим образом:

```
suit (peter, X): -
   big_salary (X,Y)),
   Y>=0.5,
```

```
good_conditions (X,Z),
 Z>=0.5.
big_salary(mailer, 0.4).
big_salary(officer, 0.7).
big_salary(bob, 0.8).
big_salary(artist, 1.0).

good_conditions(mailer, 0.7).
good_conditions(officer, 0.3).
good _conditions(bob, 0.6).
good_conditions(artist, 0.9).
```

Как видим, изменения в этом случае в тексте программы минимальные.

Теперь рассмотрим второй подход: *поиск ветви с наибольшим значением степени истинности.* Для реализации второго подхода нам потребуется воспользоваться рекурсией следующим образом:

```
database
  job(string)

suit (peter, R): -
   big_salary (X,Y)),
   good_conditions (X,Z),
    T=Y*Z.
     T>R,
   retractall(_), !,
   assert(job(X)),
   suit(peter,T).

  suit (peter, _): -
    job(X),
    write ("VYBRANA RABOTA:",X).

   big_salary(mailer, 0.4).
```

```
big_salary(officer, 0.7).
big_salary(bob, 0.8).
big_salary(artist, 1.0).

good_conditions(mailer, 0.7).
good_conditions(officer, 0.3).
good _conditions(bob, 0.6).
```

Здесь мы используем предикат базы данных job, в котором хранится в конце-концов выбранная работа. Цель данной программы должна быть записана в следующем виде

```
goal
suit(peter,0).
```

Объясним наиболее сложный участок данной программы:

```
suit (peter, R): -
   big_salary (X,Y)),
   good_conditions (X,Z),
    T=Y*Z.
     T>R,
   retractall(_), !,
   assert(job(X)),
   suit(peter,T).
```

Сначала последовательно выполняются предикаты:

```
   big_salary (X,Y)),
   good_conditions (X,Z),
    T=Y*Z.
```

Здесь по порядку выбирается работа, а затем выбирается соответствующая ей зарплата и условия. Вычисляется величина T=Y*Z. После этого выполняется проверка

$$T > R, \qquad (10.1)$$

которая в случае удачи вызовет смену содержимого предиката базы данных job, записав в него выбранную работу и снова вызвав предикат suit(peter, T), где T – новая оценка степени истинности правила. Заметим, что хотя бы одна работа всегда будет выбрана. Если нет уже ни одной работы, для которой выполняется условие (10.1), то осуществляется выход в правило:

```
suit (peter, _): -
     job(X),
     write ("VYBRANA RABOTA:",X).
```

Для вывода найденной работы на экран.

Более интересным является использование механизмов Пролога и метода многокритериального выбора Саати. Иначе говоря, в Прологе следует реализовать механизм Саати-оценки альтернатив. Рассмотрим следующий пример.

```
like(mary, X, Z):-
  rich(X, Z1),
  young(X, Z2),
  Z =A1*Z1+A2*Z2.

rich(peter, 0.7).
rich(mike, 0.5).
rich(jim, 0.3).

young (peter, 0.3).
young (mike, 0.6).
young (jim, 0.9).
```

Для решения задачи должны быть известны веса критериев A1, A2. При этом мы хотим отобрать вариант с максимальным значением итоговой оценки. Этот вариант поиска нами рассмотрен выше. Так что единственно, что привнесено в эти последние рассуждения, – это

использование Саати-оценки.

Рассмотрим, как в Прологе реализовать выбор значения функции. Это иллюстрируется следующим текстом.

```
fun(X,Y):-
  funMin(X,Y1),
  funMax(X,Y2),
Y=0.5*(Y1+Y2).

funMin(X,Y1):-
 znach(X1,Y1),
X1>=X.
funMax(X,Y1):-
  znach(X1,Y1),
X1>=X-1.

znach(0,1).
znach(1,1).
znach(2,4).
znach(3,9).
znach(4,16).
znach(5,25).
znach(6,36).
znach(7,49).
znach(8,64).
```

Найдем fun(7.5,Y).

Сначала считается funMin(7.5,Y1). Оно использует znach(8,64) и возвращает значение Y1=64. Затем считается funMax(7.5,Y2). Это вычисление полагает

X1=7.5-1=6.5 и затем считает funMin(6.5,Y2), что дает Y2=49. Затем возвращается среднее значение: 0.5*(64+49)=56.5.

Описанный способ вычисления функции будет тем точнее, чем меньше разница между соседним значениями x в списке значений znach.

ЛЕКЦИЯ 11

Принятие решений в условиях риска и неопределенности

В условиях неопределенности экспертная система или иная интеллектуальная система должна сделать выбор или последовательность выборов из совокупности различных возможных действий, исход которых носит вероятностный (неопределенный) характер. Принципы такого выбора назовем стратегией решения задачи в условиях неопределенности. Для формализации задачи введем понятие лотереи – как модели простейшей задачи выбора решения в условиях неопределенности.

Лотерея характеризуется набором исходов $Out_1, Out_2, ..., Out_z$. Для каждого исхода известна вероятность p_i и ожидаемый доход G_i (прибыль/убыток). Это можно представить таким образом (рис. 11.1):

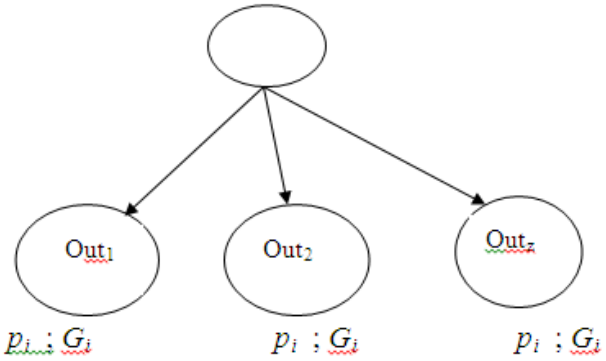

Рис. 11.1. Лотерея с набором исходов

При этом $\sum p_i = 1$.

Сыграть в лотерею – значит купить ее, но какую прибыль она принесет заранее неизвестно. Известно только, что реализуется

один из исходов $Out_1, Out_2, ..., Out_z$. Проблема выбора состоит в том, что перед лицом, принимающим решение, может стоять задача выбора какой-то одной из нескольких лотерей, либо ни одной. Рассмотрим пример. Пусть человек решает, брать ему зонтик или нет. Здесь две лотереи. Первая – брать зонтик. Вторая – не брать зонтик. Рассмотрим каждую из них.

Лотерея «брать зонт» имеет следующую схему (рис. 11.2):

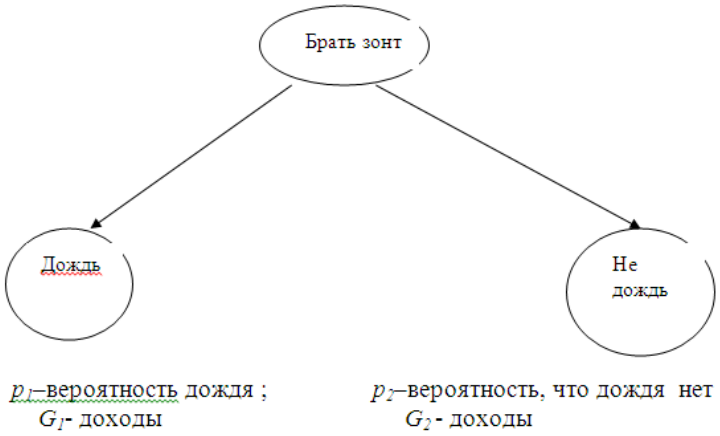

Рис. 11.2. Лотерея «брать зонт»

В данной лотерее два исхода – **дождь будет** (Out_1) и **дождя не будет** (Out_2). Пусть вероятности этих событий известны и равны соответственно: $p_1 = 0.2$, $p_2 = 0.8$. Труднее оценить доходы (потери). Для этой цели в практических задачах применяют функцию полезности F. Эта функция безразмерная и принимает значения в интервале от 0 до 1 включительно. Значение 1 соответствует наилучшему исходу, значение 0 – наихудшему. Точка 0.5 соответствует точке безразличия (индифферентности). Значения функции полезности выше 0.5 считаются доходными (благоприятствующими), а ниже 0.5 – расходными (неблагоприятствующими). В данной модели зонтик взят. Следовательно, при наличии дождя можно принять F = 1 (зонтик себя полностью оправдал). При отсутствии дождя F = 0

(зонтик взят бессмысленно, пришлось его напрасно таскать). Здесь следует заметить, что функция полезности F определяется субъективно именно тем лицом, которое и собирается брать или не брать зонтик. Следовательно, значения функции полезности для разных лиц, принимающих решение, вообще говоря, будут различными.

Теперь мы подошли к самому важному моменту – оценке лотереи.

Определение. (Объективной) Ценой лотереи называется величина:

$$O = \sum p_i G_i \ . \qquad (11.1)$$

В нашем примере O = 0.8*0 + 0.2*1 = 0.2

Цена лотереи и определяет, нужно ли принимать данное решение или нет. Полученная цена ниже 0.5 (по шкале измерения функции полезности), т.е. указывает, что данная лотерея в целом ведет к неблагоприятному исходу. Но для окончательного принятия решения нужно рассмотреть вторую лотерею из рассматриваемой задачи «не брать зонтик» (рис. 11.3).

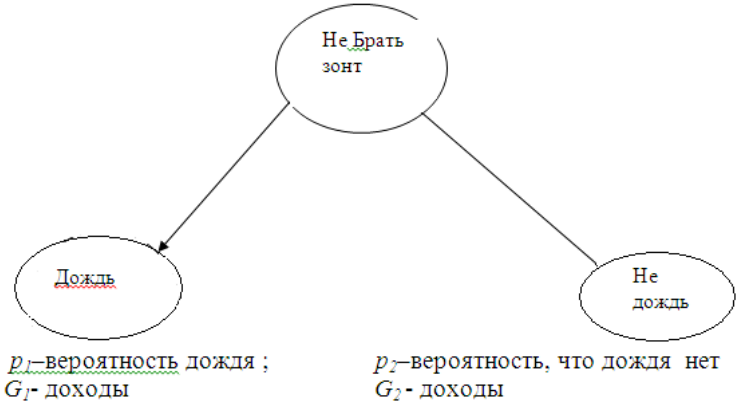

Рис. 11.3. Лотерея «не брать зонт»

В этой лотерее $G_1 = 0$ (пошел дождь, зонта не взяли), $G_2 = 1$. Цена лотереи составляет $O = 0*0.2+1*0.8 = 0.8$.

Итак, вторая лотерея сулит нам лучший результат. Вывод – при данных вероятностях и функции полезности лучше зонта не брать. Вместе с тем, ситуация может радикально измениться, если промокнуть для нас значит значительно больше, чем «бесполезно протаскать» зонт. Так, в первой лотерее зададим $G_2 = 0.5$. Оценка первой лотереи станет

$$O = 0.8*0.5 + 0.2*1 = 0.60.$$

Во второй лотерее зададим также $G_2 = 0.7$. Оценка второй лотереи теперь составит $O = 0*0.2+0.8*0.7 = 0.56$. Теперь выгоднее брать зонт.

Основными двумя вопросами являются измерение функции полезности и оценка вероятности. Обе эти задачи могут решаться в условиях неопределенности. Начнем с задания функции полезности. Этот процесс основан на методе **Саати**. В этом методе сначала нужно указать **критерии**, по которым производится оценка состояний задачи. Обычно в качестве критериев для экономических задач выступают всего два: доходы (прибыли) и расходы (убытки). Сначала мы вкладываем средства (в нечто) и несем расходы, но затем получаем прибыли (окупаем расходы). Для задачи о зонтике доходы – это сухая одежда, расходы – это затраты на таскание зонтика. Итак, имеем два очевидных критерия:

K_1 – обеспечиваем одежду сухой;
K_2 – затраты на таскание зонтика.

Далее составляется матрица критериев (рис. 11.4):

	K_1	K_2
K_1	1	4
K_2	0.25	1

Рис. 11.4

В этой матрице в ячейках записаны относительные значения приоритетов критериев по отношению друг к другу. Числа берут из шкалы Саати:

1,2,3,4,…,10, 1, $\frac{1}{10}, \frac{1}{9}, \frac{1}{8}, ..., \frac{1}{2}$. Иначе говоря, если в ячейке (i,j) записано число z из шкалы Саати, то в симметричной ей ячейке (j,i) записывается число $\frac{1}{z}$.

Числа выбирают из следующих «практических» соображений. Если критерии примерно равноценны (равнозначны), то пишем в ячейке 1. Если критерий K_i незначительно важнее критерия K_j, то пишем в ячейке (i, j) 2 или 3, а в ячейке (j,i) соответственно $\frac{1}{2}$ или $\frac{1}{3}$. При выставлении степени предпочтительности одного критерия по отношению к другому можно использовать ассоциацию с футбольным матчем. Скажем, победа со счетом 4:1 указывает на большое преимущество победителя, а победа со счетом 8:1 на подавляющее преимущество. В нашем примере мы видим, что обеспечить одежду сухой существенно важнее, чем протаскать зонтик.

После того, как матрица Саати составлена, находят произведения чисел в строках и из полученных произведений извлекают корень, степень которого равна числу критериев (в нашем примере извлекаем корень второй степени). Получим следующие значения (рис. 11.5):

	K_1	K_2	$\sqrt{произведения}$	Приоритеты α
K_1	1	4	2	$\frac{2}{2.5} = 0.8$
K_2	0.25	1	0.5	$\frac{0.5}{2.5} = 0.2$

Сумма			2.5	

Рис. 11.5. Расчет приоритетов критериев

Подсчитываем сумму найденных корней – 2.5. Теперь выполняем заключительное действие – отыскиваем приоритеты критериев путем деления найденных значений корней на сумму корней:

$\alpha_1 = 0.8, \alpha_2 = 0.2.$

Чтобы использовать эти оценки строим далее функции полезности для критериев. Функции полезности, как ранее указывалось, имеют субъективный характер. Они могут быть дискретными или непрерывными. В нашем примере построим такие функции полезности. По критерию K_1 – обеспечить одежду сухой (рис. 11.6):

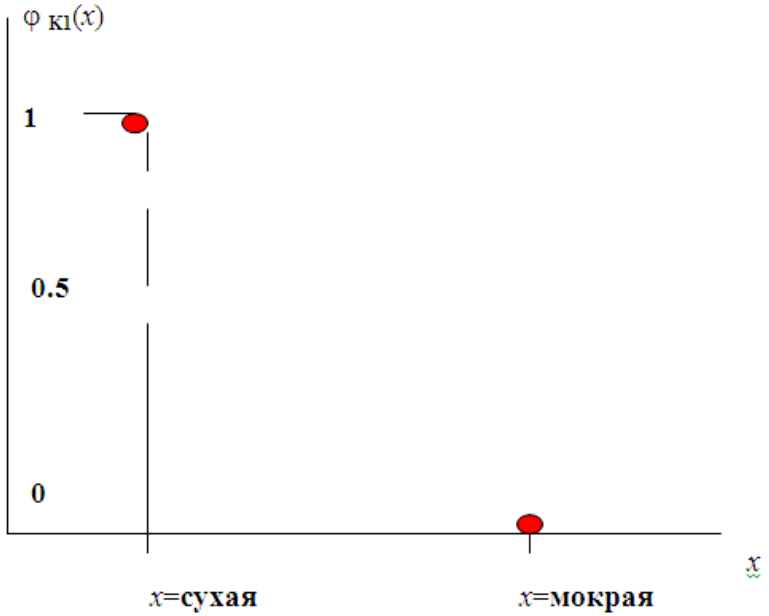

Рис. 11.6.Функция полезности по критерию K_1

По критерию K_2 – затраты на таскание зонтика (рис. 11.7):

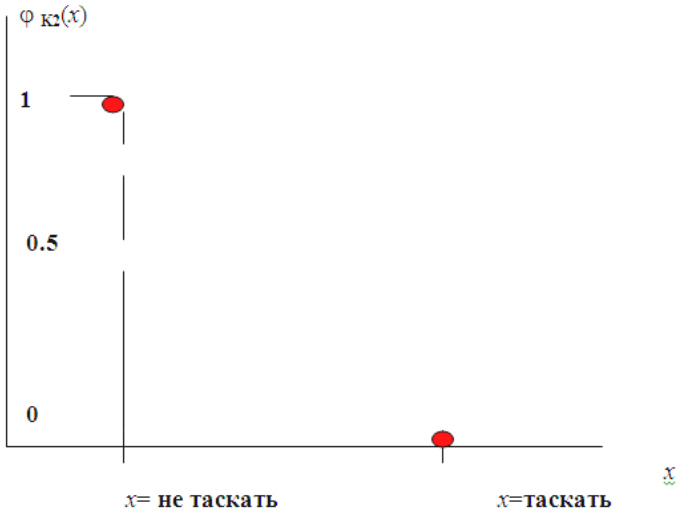

Рис. 11.7. Функция полезности по критерию K_2

Итак, наши графики функций полезности весьма элементарные. Теперь пусть вероятность дождя остается, как и ранее равной 0.2. Снова рассмотрим первую лотерею (рис. 11.8).

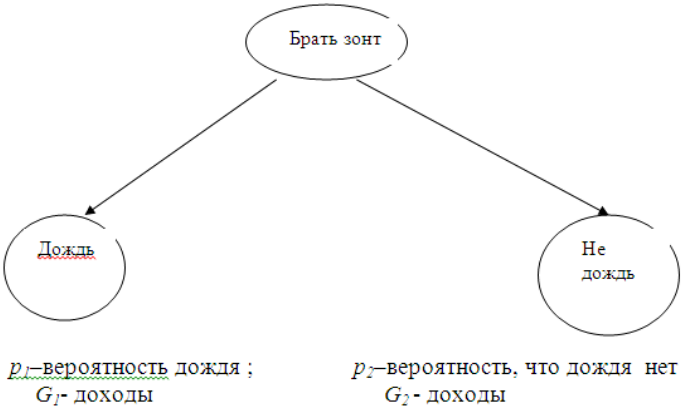

Рис. 11.8

Рассчитаем

$$G_1 = \alpha_1 \cdot \varphi_{K_1}(сухая) + \alpha_2 \cdot \varphi_{K_2}(таскать) = 0.8 \cdot 1 + 0.2 \cdot 0 = 0.8$$

Рассчитаем

$$G_2 = \alpha_1 \cdot \varphi_{K_1}(сухая) + \alpha_2 \cdot \varphi_{K_2}(таскать) = 0.8 \cdot 1 + 0.2 \cdot 0 = 0.8$$

Обратим внимание на то, что в обоих исходах одежда остается сухой и мы таскаем зонтик, так что оценки полезности совпадут. Общая оценка лотереи составит:

$$O = 0.2*0.8+0.8*0.8=0.8.$$

Для второй лотереи получим по аналогии (рис. 11.9):

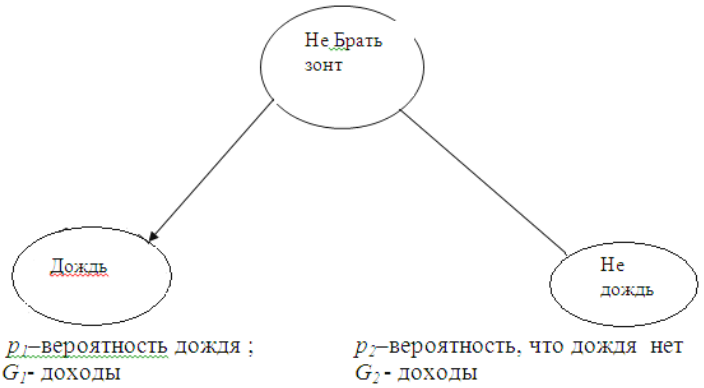

Рис. 11.9

Рассчитаем

$$G_1 = \alpha_1 \cdot \varphi_{K_1}(мокрая) + \alpha_2 \cdot \varphi_{K_2}(не\_таскать) = 0.8 \cdot 0 + 0.2 \cdot 1 = 0.2$$

Рассчитаем

$$G_2 = \alpha_1 \cdot \varphi_{K_1}(сухая) + \alpha_2 \cdot \varphi_{K_2}(не\_таскать) = 0.8 \cdot 1 + 0.2 \cdot 1 = 1$$

Оценка лотереи составит:

$$O = 0.2*0.2+1*0.8=0.84$$

Эта оценка весьма незначительно превосходит первую лотерею, так что зонтик при этом раскладе можно как брать, так и не брать (результат одинаковый). Однако, при вероятности дождя 0.3 ситуация меняется на противоположную:

$$O(\text{брать зонт}) = 0.3*0.8+0.7*0.8=0.8$$

$$O(\text{не брать зонт}) = 0.3*0.2+1*0.7=0.76$$

Теперь рассмотрим расчет вероятности. Достаточно общим слу-

чаем является применение теоремы Байеса. Это применение основано на использовании имеющихся опытных статистических таблиц. Пусть дана такая табл. 11.1.

Таблица 11.1

Фактор	Дождь (число случаев)	Не дождь (число случаев)
Всего	18	42
Пасмурно	18	42
Ветрено	5	12
Не ветрено	13	30

Пусть сегодня пасмурно (E1) и не ветрено (E2). Заметим, что все дни наблюдений были пасмурными, поскольку в непасмурный день дождей нет. Всего наблюдалось 60 дней. Определим из этой таблицы вероятность дождя, полагая, что E1 и E2 независимы. По теореме Байеса получим:

$$P(\text{дождь}/\ \text{пасмурно, неветрено}) = \frac{P(\text{дождь}) * P(\text{пасмурно}/\ \text{дождь}) * P(\text{неветрено}/\ \text{дождь})}{P(\text{дождь}) * P(\text{пасмурно}/\ \text{дождь}) * P(\text{неветрено}/\ \text{дождь}) +}$$

$$+ P(\text{недождь}) * P(\text{пасмурно}/\ \text{недождь}) * P(\text{неветрено}/\ \text{недождь})$$

Из таблицы найдем:

$$P(\text{дождь}) = \frac{18}{18+42} = 0.3$$

$$P(\text{пасмурно}/\ \text{дождь}) = 1$$

$$P(\text{неветрено}/\ \text{дождь}) = \frac{13}{18} \approx 0.7$$

$$P(\text{недождь}) = \frac{42}{60} = 0.7$$

$$P(\text{пасмурно}/\ \text{недождь}) = \frac{42}{42} = 1$$

$$P(\text{неветрено}/\ \text{недождь}) = \frac{30}{42} \approx 0.75$$

Здесь в пояснении нуждаются последние две строки. Например, как получена оценка

$$P(\textit{пасмурно}/\textit{недождь}) = \frac{42}{42} = 1$$

Дождя не было в 42 случаях. Но пасмурно было всегда, и при дожде и не при нем. Аналогично имеем

$$P(\textit{неветрено}/\textit{недождь}) = \frac{30}{42} \approx 0.75$$

В самом деле, дождя не было в 42 случаях. При этом неветрено было в 30 случаях.

Теперь получим окончательно

$$P(\textit{дождь}/\textit{пасмурно}, \textit{неветрено}) = \frac{P(\textit{дождь}) * P(\textit{пасмурно}/\textit{дождь}) * P(\textit{неветрено}/\textit{дождь})}{P(\textit{дождь}) * P(\textit{пасмурно}/\textit{дождь}) * P(\textit{неветрено}/\textit{дождь}) +}$$

$$+ P(\textit{недождь}) * P(\textit{пасмурно}/\textit{недождь}) * P(\textit{неветрено}/\textit{недождь}) =$$

$$= \frac{0.3 * 1 * 0.7}{0.3 * 1 * 0.7 + 0.7 * 0.75 * 1} \approx 0.47$$

Итак, при E1, E2 вероятность дождя 0.47.

ЛЕКЦИЯ 12

Генетические алгоритмы

В основе генетических (эволюционных) алгоритмов лежит простая идея: берут пару наилучших образцов из порожденной совокупности объектов и порождают их потомка путем взаимного обмена характеристиками (этот механизм называется *кроссовером*). Потомков порождают все время из элитных представителей текущей совокупности. Когда наступает момент и не удается улучшить требуемый функционал, производят мутацию, т.е. порождают новые объекты не в результате «скрещивания» родительских объектов, а путем случайных изменений. Эта процедура ведется до тех пор, пока удается улучшить

характеристики функционала.

Пример. Пусть требуется максимизировать функцию

$$\begin{aligned} F(x,y) &= 2\cdot x + 4\cdot y - x\cdot y, \\ x + y &\le 11 \\ x &\ge 0 \\ y &\ge 0 \end{aligned} \qquad (12.1)$$

Сформируем начальную популяцию случайным образом (рис.12.1)

№	*x*	*y*	*F(x,y)*
1	0	1	4
2	1	0	2
3	2	3	10
4	4	2	8
5	0	5	20
6	1	1	5
7	5	5	5
8	2	9	22
9	6	1	10

Рис. 12.1. Начальная популяция

Выберем элиту из четырех наилучших образцов, где функция достигает наибольших значений (рис. 12.2):

№	*x*	*y*	*F(x,y)*
3	2	3	10
5	0	5	20
8	2	9	22
9	6	1	10

Рис. 12.2

Породим потомков, например, «скрестим» пятый и восьмой объекты путем обмены координат; получим, например:

<0,9> и <2,5> при этом смотрим, чтобы выполнялось ограничение $x + y \le 11$. Таблица примет такой вид (рис. 12.3):

№	*x*	*y*	*F(x,y)*
3	2	3	10
5	0	5	20
8	2	9	22
9	6	1	10
10	0	9	36
11	2	5	14

Рис. 12.3

Видим, что лишь один потомок дал существенный прирост функции (12.1). Плохих потомков следует отбросить. Будем манипулировать данной популяцией. Скрестим десятый и восьмой объекты. Однако нам не удалось улучшить целевую функцию путем скрещивания. Более лучших значений функции при других значениях x и y мы не достигли, следовательно, прибегнем к мутации.

Мутация – небольшие произвольные изменения характеристик совокупности.

Осуществим мутацию наилучших двух образцов. Породим объекты:<2,10>,<0,10>. Таблица примет такой вид (рис. 12.4):

№	*x*	*y*	*F(x,y)*
3	2	3	10
5	0	5	20
8	2	9	22
9	6	1	10
10	0	9	36
11	2	5	14
12	2	10	24
13	10	10	40

Рис.12.4

Таким образом, удалось улучшить функционал. Снова возобновляем скрещивания и т.д., и т.п. Процесс завершаем, когда ни скрещивание, ни мутация не улучшают функционал.

ЛЕКЦИЯ 13

Неклассические логики

Понятие истины является фундаментальным в естествознании. Под **истиной** следует понимать соответствие того, что утверждается, тому, что есть в действительности. Здесь имеется два момента. Первый. Как проверить соответствие и как быть, если такая проверка невозможна в принципе? Второй. О какой *действительности* идет речь? Начнем со второго момента. Еще Аристотель указал на неоднозначность логической интерпретации утверждений о будущих событиях. Например, рассмотрим утверждение «*Завтра состоится морское сражение*». Это утверждение относится не к тому, что есть, а к тому, что будет (к будущему состоянию мира). Вместе с тем будущее состояние мира в данный момент нельзя *знать* точно, например, в силу фундаментального закона неопределенности Гейзенберга. Как же можно вести речь о соответствии тому, что точно *не определено* и принципиально не может быть точно определено в момент высказывания? Иначе говоря, вариантов будущих действительностей множество, а какой из этих вариантов реализуется заранее неизвестно. Поэтому говоря о будущих событиях, мы не знаем о какой действительности идет речь, а имеем в виду любую действительность, что нарушает трактовку истины в традиционном смысле (мы считаем, что действительность единственна).

Еще более запутана ситуация в математике, где «действительностью» является мир формальных объектов, например, прямых и точек. Так, геометрия Лобачевского исключает постулат о параллельных в геометрии Евклида, следовательно, эти две геометрии «рассматривают» разные действительности. Обратим внимание на тот принципиальный момент, что математика имеет

дело с абстрактными понятиями (например, понятием числа или точки, или бесконечно малой величины и т.п.), поэтому математических действительностей, вообще говоря, бесчисленное множество (в отличие от физической действительности).

Неопределенность в отношении к действительности связывается не только с будущим, но и с недостаточностью знаний о ней в настоящем. Например, некто утверждает, что в разложении числа π имеется бесконечное число нулей. Проверка этого утверждения, вообще говоря, может потребовать получить бесконечное разложение числа π, что невозможно. Поэтому данное высказывание неопределенно в силу незнания. Итак, отношение высказывания к действительности может иметь неопределенное значение, если действительность не определена или нельзя установить соответствие между тем, что утверждается, и тем, что есть в действительности в силу незнания.

Аристотель фактически первым поставил вопрос о необходимости измерения истины с помощью шкалы, содержащей более двух значений. Польский математик Ян Лукасевич разработал теоретическую основу сначала трехзначных, а затем многозначных логик (20-е годы прошлого века). Отметим также работы русского математика Николая Васильева (Казанский университет), который построил логическое исчисление для противоречивых формул. Вместе с тем следует отметить, что работы Лукасевича не сразу нашли признание в научном сообществе. Еще в тридцатые годы прошлого века значительная часть ученых противилась «многозначной» трактовке истины. Ясно, что в отношении физической действительности можно делать упор на ее единственность, так что утверждение «*завтра состоится морское сражение*» определенно либо будет иметь место, либо нет в той единственной действительности, которая будет завтра. Однако и в физическом мире все не так очевидно даже и без будущих событий. Например, некто говорит «Я умный». Это утверждение может быть истинным в глазах этого некто и ложным в глазах другого некто. Почему так? Потому что каждый человек видит мир своими глазами, т.е. действительностей столько, сколько точек зрения. Отсюда, в частности, следует, что в истории много утверждений

измеряются субъективно, не имеют объективной оценки.

В математике дело «сложилось» еще более запутанно. Здесь столкнулись с *парадоксами*. Парадоксы – это утверждения, которым нельзя приписать ни истинное, ни ложное значение. Именно парадоксы объективно мотивируют введение многозначных логик. Перейдем к их рассмотрению.

Логические парадоксы и их причины

Наиболее известный парадокс – это парадокс лжеца. Пусть некто заявляет: «*Я лгу*». Спрашивается, говорит ли этот некто правду или лжет. Простой анализ показывает, что приведенное утверждение не может быть ни истинным, ни ложным. Так, если оно истинно, то некто лжет в действительности, т.е. утверждение должно быть ложным. И наоборот, если утверждение ложно, то некто говорит правду, что также невозможно в силу характера утверждения.

Причина парадокса в недопустимой самоприменимости формулы к самой себе. Самоприменимость достаточно интересное явление. Например, нельзя себя вытащить за волосы из болота. Любая физическая система не может самой себе сообщить дополнительную энергию. Это случаи невозможной самоприменимости. Вместе с тем есть допустимая самоприменимость. Например, всякую программу (алгоритм) можно применить к самой себе, т.е. подать на вход программы текст этой же программы. Реакция может быть разной – от отторжения до попытки что-то посчитать.

Другой знаменитый парадокс – это парадокс «брадобрея». Он звучит так. В одном швейцарском кантоне брадобрей бреет тех и только тех, кто не бреется сам. Спрашивается, бреет ли брадобрей самого себя?

Идеи неклассических исчислений

Любое логическое исчисление с числом значений истинности, большим 2, является *неклассическим*. Так, Я. Лукасевич первоначально сформулировал исчисление с тремя значениями истинности –

0, 1, 0.5 (неопределенность). Разумеется, можно к неклассическим отнести и исчисления с двумя значениями истинности, но *иным* определением правил вывода. Такого рода исчисления мы не рассматриваем.

Итак, для каждой логической формулы f в неклассическом исчислении должна быть представлена мера истинности этой формулы в любой интерпретации (т.е. при любых значениях аргументов). Обозначим такую меру как $val(f)$.

Каковы бы ни были неклассические логики, для них справедливы следующие аксиомы.

A1. $val(false)=0$; $val(true)=1$; $1 \geq val(f) \geq 0$.
A2. Если $\alpha \rightarrow \beta$, то $val(\alpha) \leq val(\beta)$. (13.1)

Эти аксиомы настолько общи, что допускают бесконечное число различных неклассических логик. Каждая неклассическая логика задает свой вариант исчисления значений логических операций.

Вероятностная логика. Основные аксиомы

Пусть P – вероятностная мера [7]. Она удовлетворяет следующему определению, в котором T – тождественно истинная, а F – тождественно ложная формула.

(A1) $0 \leq P(f) \leq 1$ для $\forall f$, f – логическая формула;
(A2) $P(T)= 1$, $P(F)= 0$;
(A3) Если $f \rightarrow g$, то $P(f) \leq P(g)$;
(A4) Если $f \,\&\, g = F$, то $P(f \vee g) = P(f) + P(g)$ и $P(f \,\&\, g) =0$.

Легко установить следующее:

$$P(f \vee g) = P(f \vee \bar{f}g) = P(f) + P(\bar{f}g) = P(f) + P(g) - P(f \,\&\, g),$$
$$P(f \,\&\, g) \geq P(f) + P(g) - 1.$$
$$P(T) = 1 = P(\bar{f} \vee g \vee \bar{g}f) = P(\bar{f} \vee g) + P(\bar{g}f),$$
$$1 = P(f \rightarrow g) + P(\bar{g} \mid f) \cdot P(f).$$

Как следствие находим

$$P(f)=\frac{1-P(f\to g)}{P(\bar{g}\mid f)}=\frac{1-P(f\to g)}{1-P(g\mid f)} \qquad (13.2)$$

В самом деле,

$$P(\bar{g}\mid f)=\frac{P(\bar{g}f)}{P(f)},\quad P(g\mid f)=\frac{P(gf)}{P(f)},$$
$$\textit{откуда}\quad P(\bar{g}\mid f)+P(g\mid f)=1.$$

Соотношение (13.2) может быть использовано при построении вероятностных выводов.

Рассмотрим следующую схему вывода

$$\begin{array}{l} P(x)=a,\\ P(x\to y)=b\\ \hline P(y)-? \end{array} \qquad (13.3)$$

Из (13.2) получим

$$a=\frac{1-b}{1-P(y\mid x)}.$$

Воспользовавшись формулой $P(y\mid x)=\dfrac{P(y)\cdot P(x\mid y)}{P(x)}$, найдем

$$P(y)=\frac{a+b-1}{P(x\mid y)}=\frac{P(x)+P(x\to y)-1}{P(x\mid y)}. \qquad (13.4)$$

Теперь следует иметь в виду, что величина $P(x|y)$ характеризует меру связи между y и x, поэтому для ее определения нужно иметь, вообще говоря, статистические таблицы. Пусть дана такая таблица наблюдений (табл. 13.1):

Таблица 13.1

x	y
1	0
1	1
1	1
1	1
0	0
0	0
0	1
0	0

Из этой таблицы непосредственно найдем $P(x|y)=0.75$. Из (13.4), например, для $P(x)=0.6$, $P(x\to y)=0.875$ получим $P(y) = 0.63$.

Замечание. Величина $P(x|y)$ как и $P(x\to y)$ являются статистически устойчивыми значениями, характеризующими связь между переменными x, y. Поэтому соотношение (13.4) надо применять на практике при заданной статистической таблице типа табл. 13.1, но вероятность $P(x)$ может быть произвольной, удовлетворяющей лишь ограничению $\dfrac{P(x)+P(x\to y)-1}{P(x|y)}\le 1$.

Имеет место также следующая аксиома.

$$P(x)=a,$$
$$P(y)=b$$

$$\overline{\qquad\qquad\qquad}$$

$$P(x\ \&\ y)\le \min(a,b),$$
$$P(x\vee y)\ge \max(a,b).$$

Вычисление вероятностей формул

Пусть A и B – пропозициональные формулы. В силу законов вероятностной логики можно записать:

$$P(A \vee B) = P(A)+P(B)-P(AB) = P(A)+P(AB)+P(\neg AB)- P(AB) = \\ =P(A)+P(\neg AB). \qquad (13.5)$$

Здесь $P(A)$ – вероятность того, что произвольная интерпретация I выполняет формулу A. Далее, если $A\&B=false$, то $P(A \vee B) = P(A) + P(B)$. В этом случае A и B называют ортогональными. Обобщением (13.5) служит

$$P(A \vee B \vee ... \vee W) = P(A)+ P(\neg AB)+ P(\neg A\neg BC)+\ldots + \\ + P(\neg A\neg B\neg C\ldots W). \qquad (13.6)$$

Ясно, что если формула A тождественно истинна, то $P(A) = 1$. Если A и B ортогональны, то

$$P(\neg AB) = P(B). \qquad (13.7)$$

Важно заметить, что сложность процедуры вычисления вероятности экспоненциально растет от размера формулы в общем случае. Следует, что вычисление каждого слагаемого в правой части вида $P(\neg A\neg B\neg C\ldots\neg R)$ можно заменить вычислением $1- P(A' \vee B' \vee C' \vee \ldots \vee R')$, где A', B', C', $\ldots$,R' получены соответственно из A, B, C, $\ldots$, R с помощью следующих правил:

1. $x_i Y(x_i \vee Z) = x_i Y$;
2. $x_i Y(\sim x_i \vee Z) = x_i YZ.$

Следовательно, вычисление $P(\neg A\neg B\neg C\ldots W)$ реализуется рекурсивно, причем на каждом шаге рекурсии пропадает как минимум одна переменная, что гарантирует конечность процедуры.

Механизм вычисления вероятностей формул можно непосредственно использовать для проверки выводимости формул.

Пример. Показать, что из формул

$$\begin{aligned} &x_1 \vee x_2 \vee x_3, \\ &\bar{x}_1 \vee x_2, \\ &\bar{x}_2 \vee x_3 \end{aligned} \qquad (13.8)$$

выводима формула x_3. Согласно технике вывода, основанной на приведении к противоречию, добавляем к (13.8) отрицание доказываемой формулы и показываем, что вероятность выполнимости построенной системы равна 0. Имеем систему

$$\begin{aligned} &x_1 \vee x_2 \vee x_3, \\ &\bar{x}_1 \vee x_2, \\ &\bar{x}_2 \vee x_3, \\ &\bar{x}_3. \end{aligned} \qquad (13.9)$$

Необходимо убедиться, что $P(\bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1\bar{x}_2 \vee x_2\bar{x}_3 \vee x_3) = 1$. Здесь $(\bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1\bar{x}_2 \vee x_2\bar{x}_3 \vee x_3)$ есть отрицание (13.9). Воспользуемся одним упрощающим вычисления приемом: расположим формулы справа налево так, чтобы каждая последующая формула содержала как можно больше переменных, индексы которых совпадают с индексами предыдущей формулы. Так, можно использовать следующую последовательность:

$$x_3, x_2\bar{x}_3, x_1\bar{x}_2, \bar{x}_1\bar{x}_2\bar{x}_3.$$

Теперь непосредственно получаем

$$\begin{aligned} &P(13.9) = \\ &= P(x_3) + P(x_2\bar{x}_3 \cdot \bar{x}_3) + P(x_1\bar{x}_2 \cdot \bar{x}_3 \cdot (\bar{x}_2 \vee x_3) + P(\bar{x}_1\bar{x}_2\bar{x}_3 \cdot \bar{x}_3 \cdot (\bar{x}_2 \vee x_3) \cdot (\bar{x}_1 \vee x_2)) = \\ &= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = 1. \end{aligned}$$

Выводимость доказана. Следует заметить, что в общем случае вычисление вероятностей формул может потребовать экспоненциальных затрат.

ЛЕКЦИЯ 14

Примеры экспертных систем (система экологического мониторинга)

В этой лекции мы рассмотрим экспертную систему экологического мониторинга (наблюдения). Эта система выполняет следующую задачу. Ведется наблюдение за концентрацией вредных примесей в атмосфере промышленной зоны. Если концентрация превышает санитарную норму, то на основании замеров этой концентрации, данных по температуре, направлению ветра, давлению и др. делается предположение о виновнике экологического нарушения.

Задача сформулирована предельно общо. Поэтому детализируем ее постановку.

Пусть на некоторой территории расположены источники промышленного загрязнения $I_1, I_2,...,I_n$, а в выделенной локальной зоне Z этой территории, произведены замеры концентрации загрязняющих веществ, представленных вектором $X = (x_1, x_2, ...,x_m)$. Допускается, что некоторые или даже все результаты измерений окажутся выше ПДК (предельно допустимых концентраций). Требуется установить виновников загрязнения окружающей среды в выделенной зоне и степень их вины.

Задачу будем решать при следующих условиях:

- известны объемы и мощности газо-воздушных выбросов каждого загрязняющего вещества из каждого источника загрязнения,
- имеется модель рассеивания вредных примесей в приземных слоях атмосферы,
- количество измерений должно быть не менее одного, причем, чем больше измерений, тем точнее будет осуществлена экологическая инспекция.

В результате будут вычислены некоторые оценки $\beta_1, \beta_2,...,\beta_n$,

определяющие систему предпочтений в установлении виновника, при этом $\sum\beta_i = 1$ и если $\beta_p > \beta_s$, то объективно вина источника I_p оценивается выше, чем I_s.

Описание проблемы и этапов ее решения

Допустим, что на территории города за счет функционирования промышленных предприятий может возникнуть n кластеров (доменов, зон) с различной степенью загрязнения, характеризующихся векторами $\omega_1, \omega_2, ..., \omega_n$, создаваемыми $I_1, I_2, ..., I_n$ источниками загрязнения. Пусть $P(\omega_i / x)$ - условная вероятность того, что наблюдаемый вектор x относится к домену ω_i. В силу теоремы Байеса получим:

$$P(\omega_i / x) = \frac{P(\omega_i) \cdot P(x|\omega_i)}{P(x)}, \qquad (14.1)$$

где $P(x)$ – вероятность фактического наблюдения вектора x с данными значениями концентраций загрязняющих веществ;

$P(\omega_i)$ – априорная вероятность того, что виновник загрязнения – домен ω_i;

$P(x|\omega_i)$ – вероятность того, что домен ω_i мог привести к появлению вектора x;

ω_i – идентификатор (обозначение) домена.

Рассматриваются следующие домены:

ω_0 – ни один из источников не является виновником;
ω_1 – 1-й источников виновен, остальные – нет;
.....
ω_m – m-й источник виновен, остальные – нет;
ω_{m+1} – 1-й и 2-й источники виновны, остальные нет;
.....
ω_n – все n источников виновны.

Введем штрафную оценку

$$r_j(x) = \sum_{i=1}^{n} L_{ij} \cdot P(\omega_i | x), \qquad (14.2)$$

где L_{ij} – штраф, который следует заплатить за ошибочную классификацию виновника I_i вместо фактического I_j.

С учетом (14.1) перепишем (14.2) в виде

$$r_j = \frac{1}{P(x)} \sum_{i=1}^{n} L_{ij} \cdot P(\omega_i) \cdot P(x | \omega_i)$$

Поскольку P(x) является общим множителем для всех виновников, то его просто отбрасывают и формула приобретает такой вид

$$r_j = \sum_{i=1}^{n} L_{ij} \cdot P(\omega_i) \cdot P(x | \omega_i)$$

Теперь, приняв $L_{ii} = 0$ и $L_{ij} = L_{ji} = 1$ (для всех $i, j, i \neq j$), получим окончательно

$$r_j = \sum_{\substack{i=1 \\ i \neq j}}^{n} P(\omega_i) \cdot P(x | \omega_i) \qquad (14.3)$$

Формула (14.3) служит основой для принятия решений. Итак, $P(\omega_i)$ – априорная оценка вероятности виновности источника i. $P(x | \omega_i)$ – вероятность или ее оценка того, что при виновности источника i измеренная концентрация составит указанное (зафиксированное) значение x.

Введя соотношение

$$\beta_i = \frac{r_i}{\sum_{i=1}^{n} r_i}, \qquad (14.4)$$

можно утверждать, что наименьшему значению β_i будет соответствовать источник с наименьшим подозрением на виновность.

Применение формулы (14.3) потребует нескольких упрощающих допущений.

Во-первых, будем считать, что при условии соблюдения технологического регламента источником I_i измерение концентрации в зоне Z не может указать на нарушение экологических норм.

Во-вторых, предельные распределения значений концентраций газо-воздушных выбросов от каждого источника загрязнения должны подчиняться многомерному нормальному закону.

Таким образом, методика расчетов сводится к определению членов формулы (14.3). Для определения множителей $P(\omega_i)$ используется техника многокритериальной оценки на основе процедуры Саати, где в качестве альтернатив рассматриваются домены ω_i, а критериями являются факторы, обусловливающие априорные значения $P(\omega_i)$. Для оценки значений $P(x|\omega_i)$ проводится серия вычислительных экспериментов, целью которых является получение математического ожидания и среднеквадратического отклонения векторов загрязнений в домене ω_i.

Последующее изложение раскрывает существо указанной методики и ее теоретико-практическое наполнение.

Оценка $P(\omega_i)$ – априорной вероятности того, что виновник загрязнения домен ω_i

Значение искомой вероятности можно получить путем математической обработки экспертных оценок специалистов с привлечением теории многокритериальных решений и функции полезности.

Значения d_{ij} частных функций полезности, присваиваемые экспертами каждому предприятию, могут располагаться в диапазоне [0, 1]. Чем d_{ij} ближе к единице, тем, по мнению эксперта, вероятнее соответствие факта нарушения j -го критерия i- м предприятием.

Для выявления возможного предприятия - нарушителя выбраны следующие критерии:

T_1 – степень устарелости технологии и оборудования на *i*-м предприятии;

T_2 – склонность к экологическим нарушениям в прошлом на *i*-м предприятии;

T_3 – соответствие характера экологического нарушения типу технологического процесса на *i*-м предприятии;

T_4 – недостаточность административных мер для повышения ответственности за экологические нарушения на *i*-м предприятии.

Для получения обобщенной, комплексной оценки вероятности по *p* критериям одновременно необходимо определить коэффициенты δ_{sj}, характеризующие значимость, приоритеты (статистические веса) каждого критерия. Для этой цели используется алгоритм Саати, по которому строится матрица приоритетов Δ (рис. 14.1):

	$\mathbf{T}_1$	$\mathbf{T}_2$	$\mathbf{T}_3$	$\mathbf{T}_4$
$\mathbf{T}_1$	1	δ_{12}	δ_{13}	δ_{14}
$\mathbf{T}_2$	δ_{21}	1	δ_{22}	δ_{24}
$\mathbf{T}_3$	δ_{31}	δ_{32}	1	δ_{34}
$\mathbf{T}_4$	δ_{41}	δ_{42}	δ_{43}	1

Рис. 14.1. Матрица приоритетов критериев

Для каждой строки находим:

$$\lambda_s = \sqrt[p]{\prod_{j=1}^{p} \delta_{sj}} \ . \qquad (14.5)$$

Откуда веса:

$$\mu_s = \frac{\lambda_s}{\sum_{s=1}^{p} \lambda_s}. \qquad (14.6)$$

Найденные значения статистических весов считаются согласованными, если выполняется условие Саати:

$$0 \le \frac{(\lambda_{max} - p)\, l(p-1)}{x} \le 0{,}2 \qquad (14.7)$$

$$R_s = \sum_{j=1}^{p} \delta_{sj} \qquad (14.8)$$

где

$$\lambda_{max} = \sum_{s=1}^{p} \mu_s \cdot R_s \ , \qquad (14.9)$$

В зависимости от размерности матрицы приоритетов находятся величины x (табл. 14.1):

Таблица 14.1

Размер матрицы	1	2	3	4	5	6	7	8	9	10
x	0	0	0,58	0,90	1,12	1,24	1,32	1,41	1,45	1,49

Обобщенную оценку вероятности виновности источника I_i можно вычислить по формуле:

$$P(\omega_i) = \sum \mu_j \cdot d_{ji} \quad . \qquad (14.10)$$

Здесь μ_j – вес критерия j. d_{ji} – значение функции полезности критерия j.

Определение векторов $/m_i/$ оценок воздействия источника I_i на окружающую среду

Каждый источник загрязнения характеризовался по следующим параметрам:

* координата X i-го источника загрязнения;
* координата Y i-го источника загрязнения;
* высота i-го источника загрязнения;
* диаметр устья устройства выброса i-го источника загрязнения;
* нижняя граница диапазона значений объема выбрасываемой газо-воздушной смеси;
* верхняя граница диапазона значений объема выбрасываемой газо-воздушной смеси;
* нижняя граница диапазона значений мощности выброса j-го загрязняющего вещества i-го источника загрязнения;
* верхняя граница диапазона значений мощности выброса j-го загрязняющего вещества i-го источника загрязнения.

Величина максимальной приземной концентрации вредных веществ от одиночного источника с круглым устьем для выброса нагретой газо-воздушной смеси при неблагоприятных метеорологических условиях, расстояния, на котором эта концентрация достигается, а также расчеты приземной концентрации в любой точке территориального прямоугольника в зависимости от координат X и Y осуществлялись по стандартной лицензионной методике ОНД-86.

Оценка $P(x/\omega_i)$, вероятности того, что источник I_i мог привести к загрязнению выделенной зоны территории

Предельные распределения значений концентраций загрязняющих веществ от каждого источника загрязнения должны подчинять-

ся многомерному нормальному закону:

$$P(x/\omega_i) = \frac{1}{(2\pi)^{\frac{m}{2}} \cdot |c_i|^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}(x-m_i)^T \cdot c_i^{-1} \cdot (x-m_i)}, \qquad (14.11)$$

где m_i – вектор математических ожиданий концентраций загрязняющих веществ от источника I_i;

m – размерность вектора x;

c_i – ковариационная матрица векторов концентраций загрязняющих веществ;

c_i^{-1} – обратная матрица c_i;

$|c_i|$ – определитель матрицы c_i.

Для определения элементов ковариационной матрицы используется соотношение:

$$c_{kl} = \left(\frac{1}{N_i} \sum_{j=1}^{N_i} x_k^j \cdot x_l^j\right) - m_k \cdot m_l . \qquad (14.12)$$

Выводы.

Разработаны теоретические основы методики определения виновников промышленного загрязнения выделенных зон территории по результатам единичных измерений качества окружающей среды.

ЛЕКЦИЯ 15

Языки и грамматики

Языки состоят из предложений. Предложения должны составляться по правилам языка. Правила языка образуют *грамматику языка*. Рассмотрим, как определить понятие целого положительного числа с помощью грамматики.

Грамматика любого языка представляет собой набор правил, с помощью которых строятся предложения этого языка. Для записи предложений языка используются символы двух типов: символы-понятия и символы-терминалы (не являющиеся понятиями). Так, определим грамматику, раскрывающую понятие целого числа со знаком.

ЧИСЛО → ЗНАК ЧИСЛО
ЧИСЛО → ЦИФРА ЧИСЛО
ЧИСЛО → ЦИФРА
ЗНАК → +
ЗНАК → -
ЦИФРА → 0|1|2|3|4|5|6|7|8|9

В этой грамматике понятиями являются ЧИСЛО, ЦИФРА, ЗНАК. Терминальными символами являются +, –, 0, 1,2,…,9. Понятия раскрываются через другие понятия и терминальные символы, в то время как терминальные символы не раскрываются через иные символы, т.е. являются конечными (первичными) объектами.

На вход программы будем подавать цепочки символов, а программа должна выяснить, является ли переданная цепочка числом, определенным согласно грамматике выше, либо нет.

Достаточно просто выполнять разбор языка в среде Пролога.

Пролог позволяет выполнить кодирование требуемой программы практически напрямую:

```
predicates
nondeterm  number(string)
nondeterm  sign(char)
nondeterm  digit(char)

clauses

      number(X):-
              frontchar(X,Y,XR),
```

```
                sign(Y),
                number(XR).

        number(X):-
                frontchar(X,Y,XR),
                digit(Y),
                number(XR).

        number(X):-
                str_char(X,C),
                digit(C).

            sign('+').
            sign('-').

            digit(R):-
R='0';R='1';R='2';R='3';R='4';R='5';R='6';R='7';R='8';R='9'.

        goal
            number("-99"),write("YES"),readchar(_);
            write("NO"),readchar(_).
```

Для проверки аналогий возьмем, например, следующее правило грамматики:

ЧИСЛО → ЗНАК ЧИСЛО

Этому правилу грамматики соответствует следующее правило (clause) языка Пролог:

```
    number(X):-
                frontchar(X,Y,XR),
                sign(Y),
                number(XR).
```

Проверке знака (sign) соответствуют правила

```
sign('+').
sign('-').
```

Правилу грамматики

ЧИСЛО → ЦИФРА ЧИСЛО

соответствует правило Пролога

```
number(X):-
        frontchar(X,Y,XR),
        digit(Y),
        number(XR).
```

и т.д.

Самыми простыми языками являются регулярные языки. Они имеют самые простые наборы правил. Для распознавания регулярных языков можно использовать регулярные автоматы. Эти автоматы являются также вариантом алгоритмического устройства, т.е. некоторым «видом» машин Тьюринга. По существу, приведенная выше программа на языке Пролог, реализовала программно *автомат*.

Удобно правила регулярного автомата представлять в виде следующих формул

$$Q_i\ a_k \rightarrow Q_j \qquad (15.1)$$

Такое правило читается следующим образом: если автомат находится в состоянии Q_i и поступает символ a_k, то автомат переходит в состояние Q_j. Подобный набор правил и определяет поведение автомата, т.е. *задает* автомат.

Для того чтобы построить таблицу разбора для регулярного автомата, достаточно каждому правилу (15.1) сопоставить фрагмент таблицы следующего вида (рис. 15.1):

. . . . .	a_k

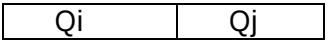

Рис. 15.1

При построении автоматов, распознающих языки, следует состояния автомата определять исходя из того, какого рода символы в этом состоянии ожидаются на входе автомата. Снова рассмотрим простой арифметический язык, но допустим, что кроме переменных языка могут встречаться и числовые константы.

Пример.

```
a+5*c
a-b+c*10-d
a
```

Пусть в состоянии Q_0 ожидается поступление буквы или числовой константы. В состоянии Q_1 ожидается поступление знака операции. В состоянии Q_2 ожидаем поступления цифры или знака операции. Отсюда получаем правила:

Q_0 **буква** → Q_1
Q_1 **знак_операции** → Q_0
Q_0 **цифра**→ Q_2
Q_2 **знак_операции** → Q_0
Q_2 **цифра** → Q_2

Вместе с тем, если ввести в язык скобки, то нельзя реализовать его распознавание с помощью регулярного автомата, поскольку нужно обеспечить распознавание парности открывающих и закрывающих скобок.

Имеет место следующий интересный результат.

Теорема. Всякий недетерминированный регулярный автомат можно реализовать с помощью эквивалентного детерминированного регулярного автомата.

Доказательство теоремы будет проведено на примере требуемой реализации.

Пусть имеется табл. 15.1 недетерминированного автомата.

Таблица 15.1

	a	b
Q0	Q1,Q2	Q2
Q1	QSTOP	Q2
Q2	Q1	QSTOP

Недетерминизм автомата определяется наличием клетки (и подобных ей в других случаях) вида (рис. 15.2):

	a
Q0	Q1,Q2

Рис.15.2

Здесь переход из состояния Q0 под действием одного и того же символа может вести в состояние Q1 или в состояние Q2. Заменим два этих состояния на новое, например, Q3={Q1,Q2}. Теперь табл. 15.1 примет такой вид:

Таблица 15.2

	a	b
Q0	Q3	Q2
Q1	QSTOP	Q2
Q2	Q1	QSTOP
Q3	QSTOP,Q1	QSTOP,Q2

Объясним содержимое строки Q3. Если под действием a переходим из Q0 в Q1, то под действием a из Q1 перейдем в QSTOP. Если под действием a перейдем из Q0 в Q2, то под действием a перейдем из Q2 в Q1. Отсюда получаем ячейку (рис. 15.3)

	A
Q3	QSTOP,Q1

Рис. 15.3

Аналогичным образом получаем ячейку в столбце b.

Теперь поступаем аналогично. Заменим состояния QSTOP,Q1 на новое – Q4 (табл. 15.3):

Таблица 15.3

	a	b
Q0	Q3	Q2
Q1	QSTOP	Q2
Q2	Q1	QSTOP
Q3	Q4	QSTOP,Q2
Q4	QSTOP	Q2

Следует просто заметить, что из QSTOP уже нет выхода. Таким образом, итоговая таблица примет следующий вид (табл. 15.4):

Таблица 15.4

	a	b
Q0	Q3	Q2
Q1	QSTOP	Q2
Q2	Q1	QSTOP
Q3	Q4	Q5
Q4	QSTOP	Q2
Q5	Q1	QSTOP

Рассмотрим, например, разбор слова aabb. По последней таблице попадаем в QSTOP. Варианты цепочек для соответствующего недетерминированного автомата таковы:

Q0-Q1-QSTOP;

Q0-Q2-Q1-Q2-QSTOP.

ЛЕКЦИЯ 16

Информационно-диагностическая экспертная медицинская система “Нефрон”

Экспертные системы (ЭС) являются одним из ведущих направлений в области искусственного интеллекта. Их основным назначением является поддержка поиска решения так называемых интеллектуальных задач (см. далее). ЭС обеспечивают необходимые механизмы для разработки программных средств, которые при решении интеллектуальных задач дают результаты, сравнимые по качеству и эффективности с решениями, даваемыми экспертами в данной области.

В России и других странах СНГ существуют экспертные системы для ортопедии, кардиологии, почечных заболеваний и пр. Их успех определяется степенью точного понимания реальных медико-технических процессов. Системы часто совмещены с мультимедийными устройствами.

Ряд интеллектуальных систем позволяют учитывать мнение врача (ДИАГЕН), реализуют механизм корректировки "весов" признаков, вводят коэффициенты "уверенности" и др. Такие системы обеспечивают варианты решений: "мягкое решение", "жесткий выбор", и др.

В практике медицины применяется технология имитационного динамического моделирования для диагностики, в частности, электро-вибростимуляции позвоночного столба человека.

Ниже приведено описание информационно-диагностической экспертной медицинской системы для диагностирования почечных заболеваний и определения склонности к хронизации болезни (гломерулонефрита). Система использует ряд встроенных механизмов и технологий, описанных ранее в этом курсе. Общая технология решения сложных задач диагностирования и принятия решений состоит в следующем.

1. Используя механизм логического вывода в системе ЕСЛИ-ТО, получаем отфильтрованное множество заболеваний, на котором предстоит поставить диагноз. Это отфильтрованное множество вы-

бирается из базы данных, в которой хранятся сведения по всем болезням, с зарегистрированными эпикризами.

2. На основе систем многокритериальной оценки поступивший больной оценивается по множеству показателей (функциональных подсистем, таких как кровь, лимфа, мочеполовая система, сердце и легкие, кожа и др.)

3. Полученная комплексная оценка используется для оценки вероятности хронизации заболевания.

Важность распознавания хронизации болезни на ранних этапах существенна при выборе схемы лечения. Таким образом, решаемая системой задача приобретает очевидный практически значимый характер.

ПРОБЛЕМА ДИАГНОСТИРОВАНИЯ В ЭС

Байесовская стратегия постановки диагнозов

Постановка диагноза – это сложный процесс, связанный с решением так называемой интеллектуальной задачи. Интеллектуальная задача характеризуется следующими признаками [8-10]:

- нечеткая постановка или неполнота исходных данных задачи;
- отсутствие точного алгоритма решения задачи;
- огромное число возможных исходов;
- невозможность формализации задачи;
- существенное использование эвристических и приближенных методов;
- наличие экспертов в области задачи.

Легко видеть, что задачи (медицинской) диагностики вполне удовлетворяют этим признакам. Поэтому для решения таких задач используют современные технологии инженерии знаний, главным образом – экспертные системы (ЭС).

Наиболее широкое распространение подобные технологии получили в тех медицинских учреждениях, где к точности постановки диагноза предъявляются повышенные требования, например, в невропатологии и нейрохирургии, сердечной и почечной патологии и т.д.

Для математического решения проблемы диагностики в настоя-

щее время наиболее широко применяют три группы методов:

- статистические;
- логические;
- нечеткие.

Каждая из этих групп методов имеет свои достоинства и недостатки. Следует заметить, что логические методы предполагают знание точных причинно-следственных связей на множестве признаков и причин заболеваний. Это является существенным ограничением, поскольку многие признаки являются размытыми или общими для нескольких заболеваний. Поэтому применение логических методов либо не всегда оправдано, либо невозможно.

Естественным поэтому является использование математических методов, использующих вероятностные или нечеткие методологические принципы.

Если рассматривать нечеткий подход к диагностике, то его целесообразность связана с теми ситуациями, когда нельзя сделать статистические выводы, например, не известно распределение, которому подчиняется тот или иной признак, либо статистика собрана в недостаточном объеме, либо собранные данные недостоверны. Такие ситуации вполне могут иметь место на практике. Однако в большинстве случаев медицинское учреждение располагает накопленными данными из анамнезов пациентов, по которым можно провести качественную диагностику, базируясь на статистических методах.

Пусть необходимо производить диагностику между заболеваниями D_1, D_2,…, D_n . Для каждого из этих заболеваний известны условные вероятности $P(S \mid D_j)$, характеризующие появление набора симптомов S при заболевании D_j. Здесь $S = \{ S_1 , S_2 ,\ldots, S_k \}$ и S_k – различные симптомы. Пусть далее $P(D_m)$ – априорная вероятность заболевания D_m. Тогда задача дифференциальной диагностики может быть сведена к статистической задаче выбора гипотез, решение которой основывается на использовании известной теоремы Байеса:

$$P(D_j \mid S) = \frac{P(D_j) \cdot P(S \mid D_j)}{\sum_{j=1}^{n} P(D_j) \cdot P(S \mid D_j)} \qquad (16.1)$$

Если для какого-нибудь диагноза D_j рассчитанная вероятность (16.1) значительно превосходит значение этой вероятности для других диагнозов, то оптимальное правило приписывает больному заболевание D_j .

Использование формулы (16.1) встречается со следующими трудностями: если априорные вероятности $P(D_m)$ можно сравнительно просто определить из базы данных пациентов медицинского учреждения, то условные вероятности $P(S \mid D_j)$ можно сравнительно легко вычислить лишь при небольшом числе признаков S_1, S_2 ,…, S_k. Во-первых, это может быть связано с тем, что число различных комбинаций признаков может быть гигантским. Например, если каждый симптом имеет только 2 различных значения (ДА/НЕТ) и число симптомов равно 30, то число всех возможных симптомокомплексов может исчисляться величиной 2^{30}. Не спасает ситуации даже столь важное предположение, что признаки являются взаимно независимыми. При этом допущении знаменатель

$$\sum_{j=1}^{n} P(D_j) \cdot P(S \mid D_j)$$

вычисляется по упрощенной формуле:

$$\sum_{j=1}^{n} P(D_j) \cdot P(S \mid D_j) = \sum_{j=1}^{n} P(D_j) \cdot P(S_1 \mid D_j) \cdot P(S_2 \mid D_j) \cdot \ldots \cdot P(S_k \mid D_j)$$

Однако и теперь может иметь место случай, когда в базе данных ранее не наблюдалась ситуация $(S_\alpha \mid D_j)$, так что получаем в знаменателе 0, и, следовательно, формула Байеса становится неприменимой.

Таким образом, перед нами стоит следующая проблема: усовершенствовать механизм диагностирования на основе байесовского разрешающего правила при следующих допущениях:

- число различных симптомов велико (десятки-сотни)
- симптомы считаются независимыми
- возможны ситуации, при которых нет сведений по симптому S_α для диагноза D_j.

Пример расчетов. Пусть по данным статистики при заболевании D_1 признак S_1 встречается в 75% случаев, признак S_2 не встречается, признак S_3 наблюдается в 10% случаев, а S_4 – в 40%; данные по встречаемости признаков при заболевании D_2 таковы: S_1 – 10%, S_2 – 10%, S_3 – 50% , S_4 – 5%; данные по встречаемости признаков при заболевании D_3 таковы: S_1 – 15%, S_2 – 10%, S_3 – 90% , S_4 – 70% .

Эти данные можно представить следующей табл. 16.1.

Таблица 16.1

D_j	$P(D_j)$	$P(S_1 \mid D_j)$	$P(S_2 \mid D_j)$	$P(S_3 \mid D_j)$	$P(S_4 \mid D_j)$
D_1	0.3	0.75	0	0.1	0.4
D_2	0.1	0.1	0.1	0.5	0.05
D_3	0.6	0.15	0.1	0.9	0.7

Пусть известно, что у пациента имеется признак S_1 и признак S_2, а признаки S_3 и S_4 отсутствуют. Считая признаки независимыми, определим вероятности диагнозов. Так, если признак S_α при диагнозе D_β отсутствует, то рассчитываем вероятность $P(\bar{S}_\alpha \mid D_\beta)$ по формуле

$$P(\bar{S}_\alpha \mid D_\beta) = 1 - P(S_\alpha \mid D_\beta) \qquad (16.2)$$

$$P(D_1 \mid S_1, S_2, \bar{S}_3, \bar{S}_4) = \frac{P(D_1) * P(S_1, S_2, \bar{S}_3, \bar{S}_4 \mid D_1)}{\sum P(D_j) \cdot * P(S_1, S_2, \bar{S}_3, \bar{S}_4 \mid D_j)} =$$

$$= \frac{P(D_1) * P(S_1 \mid D_1) * P(S_2 \mid D_1) * P(\bar{S}_3 \mid D_1) * P(\bar{S}_4 \mid D_1)}{\sum P(D_j) * P(S_1, S_2, \bar{S}_3, \bar{S}_4 \mid D_j)} =$$

$$\frac{P(D_1) * P(S_1 \mid D_1) * P(S_2 \mid D_1) * (1 - P(S_3 \mid D_1)) * (1 - P(S_4 \mid D_1))}{\sum P(D_j) * P(S_1, S_2, \bar{S}_3, \bar{S}_4 \mid D_j)}$$

Непосредственно определяем

$$P(D_1 \mid S_1, S_2, \bar{S}_3, \bar{S}_4) = \frac{0.3 * 0.75 * 0 * (1 - 0.1) * (1 - 0.4)}{0.3 * 0.75 * 0 * (1 - 0.1) * (1 - 0.4) + 0.1 * 0.1 * 0.1 * (1 - 0.5) * (1 - 0.05) + + 0.6 * 0.15 * 0.1 * (1 - 0.9) * (1 - 0.7)} = 0$$

$$P(D_2 \mid S_1, S_2, \bar{S}_3, \bar{S}_4) = \frac{0.1 * 0.1 * 0.1 * (1 - 0.5) * (1 - 0.05)}{0.3 * 0.75 * 0 * (1 - 0.1) * (1 - 0.4) + 0.1 * 0.1 * 0.1 * (1 - 0.5) * (1 - 0.05) + + 0.6 * 0.15 * 0.1 * (1 - 0.9) * (1 - 0.7)} = 0.65$$

$$P(D_3 \mid S_1, S_2, \bar{S}_3, \bar{S}_4) = \frac{0.6 * 0.15 * 0.1 * (1 - 0.9) * (1 - 0.7)}{0.3 * 0.75 * 0 * (1 - 0.1) * (1 - 0.4) + 0.1 * 0.1 * 0.1 * (1 - 0.5) * (1 - 0.05) + + 0.6 * 0.15 * 0.1 * (1 - 0.9) * (1 - 0.7)} = 0.35$$

Таким образом, следует выдвинуть гипотезу как о наиболее вероятном диагнозе D_2. В данном примере рассматривается пространство элементарных событий $\{D_1, D_2, D_3\}$, любые два из которых не могут произойти одновременно и одно из этих событий непременно имеет место. Это является упрощающим допущением,

поскольку возможны в принципе исходы с двумя и более одновременно присутствующими заболеваниями, а также исход, в котором нет ни одного из указанных диагнозов. Следовательно, в общем случае подлежат расчету следующие вероятности:

$$P(D_1\overline{D}_2\overline{D}_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4)$$
$$P(\overline{D}_1 D_2\overline{D}_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4)$$
$$P(\overline{D}_1\overline{D}_2 D_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4)$$
$$P(\overline{D}_1\overline{D}_2\overline{D}_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4)$$
$$P(D_1 D_2\overline{D}_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4)$$
$$P(D_1\overline{D}_2 D_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4)$$
$$P(\overline{D}_1 D_2 D_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4)$$
$$P(D_1 D_2 D_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4)$$

Опять же и в этом случае используется упрощающее допущение, что диагнозы не зависят друг от друга, так что, например, допускаем правомерным использование следующего расчетного соотношения:

$$P(D_1 D_2\overline{D}_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4) =$$

$$= P(D_1 \mid S_1, S_2, \overline{S}_3, \overline{S}_4) \cdot P(D_2 \mid S_1, S_2, \overline{S}_3, \overline{S}_4) \times \qquad (16.3)$$

$$\times P(\overline{D}_3 \mid S_1, S_2, \overline{S}_3, \overline{S}_4).$$

На практике, однако, формула (16.3) не находит применения, поскольку при постановке диагноза эксперта интересует, верно ли предположение о конкретном диагнозе (заболевании) вне зависимости от его сочетания с другими возможными заболеваниями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Герман, О.В. Экспертные системы: учебно-методическое пособие / О.В. Герман. – Минск: БГУИР, 2008. – 91 с.
2. Герман, О.В. Экспертные системы: лабораторный практикум / О.В. Герман, Н.В. Батин. – Минск: БГУИР, 2003.
3. Чень, Ч. Математическая логика и автоматическое доказательство теорем / Ч. Чень, Р. Ли. – М.: Наука, 1983. – 358 с.
4. Гэри, М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон. – М.: Мир, 1982. – 600 с.
5. Дюбуа, Д. Теория возможностей / Д. Дюбуа, А. Прад. – М.: Радио и связь, 1990. – 288 с.
6. Герман, О.В. Система вывода для нечеткой логики на основе многозначных исчислений Лукасевича / О.В. Герман, И.Г. Блохина, Ю.О. Герман // Актуальные проблемы гуманитарных и естественных наук. – 2010. – № 6. – С. 35–38.
7. Гиндикин, С.Г. Алгебра логики в задачах / С.Г. Гиндикин. – М.: Наука, 1972. – 286 с.
8. Уотермен, Д. Руководство по экспертным системам / Д. Уотермен. – М.: Мир, 1989.
9. Построение экспертных систем / под ред Ф. Хейес-Рота. – М.: Мир, 1987.
10. Герман, О.В. Введение в теорию экспертных систем и обработку знаний / О.В. Герман. – Минск: Дизайн-Про, 1995. – 256 с.

Учебное издание

ГЕРМАН Олег Витольдович
ГЕРМАН Юлия Олеговна

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Методическое пособие
для студентов специализации 1-40 01 02-01 «Информационные системы и технологии в проектировании и производстве»

Технический редактор О.В. Песенько
Компьютерная верстка А.Г. Занкевич

Подписано в печать 31.01.2012.
Формат 60×84 1/16. Бумага офсетная.
Отпечатано на ризографе. Гарнитура Таймс.
Усл. печ. л. 7,38. Уч.-изд. л. 5,77. Тираж 100. Заказ 1005.

Издатель и полиграфическое исполнение:
Белорусский национальный технический университет.
ЛИ № 02330/0494349 от 16.03.2009.
Проспект Независимости, 65. 220013, Минск.