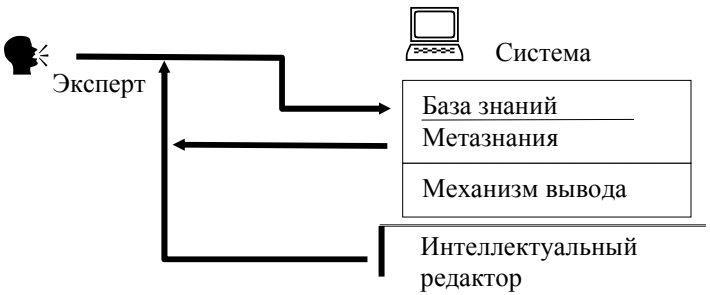
Томский межвузовский центр дистанционного образования

И.А. Ходашинский

МЕТОДЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА, БАЗЫ ЗНАНИЙ, ЭКСПЕРТНЫЕ СИСТЕМЫ

Учебное пособие



ТОМСК – 2002

Министерство образования Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

И.А. Ходашинский

МЕТОДЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА, БАЗЫ ЗНАНИЙ, ЭКСПЕРТНЫЕ СИСТЕМЫ

Учебное пособие

2002

Корректор: Красовская Е.Н.

Ходашинский И.А.
Методы искусственного интеллекта, базы знаний, экспертные системы: Учебное пособие. – Томск: Томский межвузовский центр дистанционного образования, 2002. – 168 с.

В пособии излагаются основные проблемы представления знаний и вывода на знаниях, а также вопросы построения экспертных систем. Рассмотрены четыре модели представления знаний: логическая, продукционная, сетевая, фреймовая. Изложены основы обработки нечетких знаний. Описаны основные положения инженерии знаний. В качестве среды реализации выбран язык ПРОЛОГ. Рассмотрены базовые структуры языка ПРОЛОГ и техника программирования на этом языке. В пособии приведены методические указания и контрольные работы по темам курса.

Пособие предназначено для студентов, изучающих основы искусственного интеллекта, и специалистов в области символьной обработки данных.

1 ПРЕДМЕТ И МЕТОДЫ ДИСЦИПЛИНЫ

1.1 История

Искусственный интеллект – научное направление, появившееся во второй половине двадцатого века на стыке таких научных дисциплин, как математика, вычислительная техника, логика, программирование, психология, лингвистика, нейрофизиология. Задачей этого научного направления является воссоздание с помощью компьютера разумных рассуждений и действий.

Интеллект, по С.И. Ожегову, – это «ум, мыслительная способность, умственное начало у человека», т.е. субстанция, присущая исключительно человеку. Однако компьютер сегодня способен доказывать теоремы, решать головоломки, переводить с одного языка на другой, сочинять стихи и музыку, диагностировать заболевания, играть в шахматы на уровне гроссмейстера и участвовать в других творческих процессах. Это свидетельствует о том, что с помощью компьютера можно автоматизировать такие виды человеческой деятельности, которые традиционно называют интеллектуальными. Однако об интеллекте компьютера можно говорить лишь тогда, когда компьютер, опираясь на собственные знания, синтезировал бы программу (спланировал действия) выполнения творческих процессов. Интеллект сводится не к умению выполнять ту или иную интеллектуальную деятельность, а к пониманию того, как научиться новому виду интеллектуальной деятельности. Различные трактовки понятия «интеллект» привели к нескольким направлениям в искусственном интеллекте.

📖 ***Сведение понятия «интеллект» к понятию «знание» породило когнитивистский подход в искусственном интеллекте. Отождествление понятий «интеллект» и «способность к обучению» породило бионическое направление.***

Искусственный интеллект (ИИ) ведет свою историю со второй половины 50-х годов. В эти годы начались дискуссии о природе процессов, происходящих в мозге, и о возможности их компьютерного моделирования. В 1956 году была продемонстрирована первая интеллектуальная программа «Логик – Теоретик», предназначенная для доказательства теорем, в 1957 г. – программа для игры в шахматы NSS. В 1956 году была проведена первая конференция по искусственному интеллекту в Джорджтаунском колледже. В эти годы специалисты по ИИ пытались моделировать процесс мышления как поиск общих методов решения широкого круга задач. Это привело к появлению концепции универсальных эвристических схем, реализованных в виде про-

грамм типа Универсального Решателя Задач (GPS). В основе построения этих программ лежал принцип уменьшения перебора альтернатив на основе здравого смысла, применения числовых функций оценивания и различных эвристик. Однако, несмотря на известный прогресс, такой подход не привел к качественному скачку в развитии ИИ. Разработка программ общего назначения оказалась слишком сложной и, в конечном счете, бесполезной задачей. Чем большее число классов задач может обрабатываться одной программой, тем с более низким качеством ею осуществляется обработка в рамках любой отдельной задачи.

В русле когнитивистского подхода на смену универсальным эвристическим схемам пришло понимание, что должны существовать некоторые другие методы и способы наделения компьютера интеллектом. Это возможно сделать, посчитали специалисты по ИИ, путем наделения программы знаниями. Здесь знания рассматриваются как множество реально существующих символов. Под символом понимается смысловой образ, которым может быть отдельное понятие, рисунок, звуковой сигнал. Задав операции над символами, можно манипулировать знаниями. К простейшим операциям над символами могут быть отнесены следующие действия: запоминание символов и отношений между ними; создание, изменение и удаление символов; сравнение символов; формирование выводов в зависимости от проведенных сравнений.

Начиная с 70-х годов специалисты по ИИ сосредоточили свои усилия на разработке методов и способов представления и использования знаний. *Представление знаний* – это ориентированная на компьютерную обработку методология моделирования и формализации концептуальных знаний. *Использование знаний* – это технология вывода или получения решения в рамках данного представления. Типичными моделями представления знаний являются: логические, продукционные, фреймовые и сетевые. Однако поиски достаточно эффективных и универсальных систем представления знаний также не привели к ощутимым результатам.

Практический интерес к интеллектуальным системам возник только с появлением экспертных систем. Это произошло в конце 70-х годов, когда специалисты в области ИИ осознали, что эффективность программы при решении задач определяется теми знаниями, которые в программу заложены, а не просто теми формальными построениями и схемами вывода, которые в ней используются. Новый принцип построения интеллектуальных систем может быть сформулирован так:

📖 ***Для того чтобы наделить программу интеллектом, ее следует снабдить большим объемом высококачественных конкретных знаний из некоторой узкой предметной области.***

1.2 Терминология

Ценность экспертной системы определяется не столько конкретными формализмами и используемыми схемами вывода, сколько теми знаниями, которыми она располагает. Центральным звеном экспертной системы является мощная база знаний. *База знаний* разрабатывается как модель ограниченной части мира, которая позволяет, задав механизм вывода, рассуждать об этом мире. Знания накапливаются в процессе построения системы, представляются в явно сформулированном виде и должны быть организованы таким образом, чтобы облегчить процесс принятия решений. Явный характер и доступность знаний отличает экспертную систему от обычных программ.

Воспользовавшись диаграммами Вена, определим место интеллектуальных систем, систем, основанных на знаниях, и экспертных систем на множестве компьютерных систем (см. рис. 1.1).

Рис. 1.1

Компьютерная система – это программно-аппаратный комплекс, построенный, в том числе, и не на базе машины фон Неймана. *Интеллектуальной* называется компьютерная система, способная выполнять действия, которые характеризуются как разумные, интеллектуальные. *Системы, основанные на знаниях*, характеризуются наличием в их составе двух компонентов: базы знаний и машины логического вывода. *Экспертные системы* относятся к классу систем, основанных на знаниях, базируются на знаниях эксперта, работают в узких предметных областях и способны объяснить свои действия и результаты.

Как научное направление ИИ имеет свой объект и свои специфические методы исследования этого объекта. Объектом изучения являются процедуры, используемые человеком при решении интеллектуальных задач с целью воссоздания компьютерных моделей этих процедур. Основные методы ИИ связаны с применением разного рода программных и теоретических моделей. В когнитивистском направлении это логические и семантические модели, в бионическом – нейронные сети и алгоритмы эволюционного моделирования.

Рассмотрим области применения искусственного интеллекта. В своей работе Лорьер утверждает, что всякая задача, для которой неизвестен алгоритм решения, априори относится к задачам искусственного интеллекта [3]. Под алгоритмом понимается вся последовательность действий, которые четко определены и выполнимы на компьютере в приемлемое время. Области применения искусственного интеллекта характеризуются следующими особенностями:

- использование информация в символьной форме – букв, слов, знаков, рисунков, что отличает область ИИ от областей традиционного использования компьютеров, где обработка ведется в числовой форме;
- наличие выбора, причем выбор делается в условиях неопределенности.

1.3 Знание. Модели

Знание, которое одно человеческое поколение передает другому, может быть условно разделено на два типа. Первый тип – общедоступные знания – это факты, сведения, определения и теории, описываемые в книгах, справочниках, учебниках, отчетах. Второй тип – индивидуальные знания – это человеческое умение решать задачи, ставить диагноз и лечить больных, сочинять музыку, находить и устранять неисправности в технике, вести анализ военных действий и т.д. Знания этого типа состоят из эмпирических правил, которые называют еще «эвристиками». Эвристики позволяют специалистам выдвигать гипотезы, находить перспективные подходы к решению задач, решать проблемы в условиях неполных и ненадежных данных. Знания первого типа могут быть получены в процессе обучения в школе, училище, университете. Знания второго типа передаются от учителя к ученику, приобретаются в результате решения многочисленных задач и связаны с большими затратами. Поэтому извлечение знаний и придание им формы, позволяющей использовать их в компьютере, может удешевить знание и сделать его доступным и воспроизводимым.

В компьютерных науках термин «знание» непосредственно связан с термином «данные» и отчасти с термином «программа». Можно сказать, что знания – это сложно структурированные данные. Данные прошли эволюционный путь от простых машинных слов с заданным числом битов до сложных динамических структур типа вектора, списка, матрицы, записи. У данных появились новые качественные свойства, которые позволили перевести их в класс знаний. Ниже рассмотрим четыре особенности, присущие этой форме представления информации в компьютере [2].

Внутренняя интерпретируемость. При переходе к знаниям память компьютера содержит информацию о некоторой метаструктуре хранимых информационных единиц. Вместе с информационной единицей, представляющей собственно элемент данных, в памяти компьютера содержится система имен, связанная с такой информационной единицей. Система имен включает в себя индивидуальное имя, присвоенное информационной единице, и ряд имен тех множеств или классов, в которые эта единица входит. Наличие системы имен позволяет системе «знать», что хранится в ее памяти, и уметь отвечать на вопросы о содержимом памяти. Подобная внутренняя интерпретация принята и в системах управления базами данных.

Структурированность. Одной из особенностей человеческого познания окружающего мира является его способность к декомпозиции наблюдаемых объектов, умение выделять в них отдельные элементы и связи между ними. Это позволяет человеку воспринимать любой объект как некоторую структуру, сложные объекты – в виде совокупности более простых и познавать свойства нового объекта через свойства тех, что входят в его структуру. На уровне представлений о мире свойство структурированности выражается в том, что соответствующие понятия оказываются соединенными между собой связями типа «род-вид», «часть-целое».

Связность. Знания могут быть связаны друг с другом не только связями, рассмотренными выше. Имеются и другие связи, например, «одновременно», «быть рядом», «выше», т.е. между информационными единицами устанавливаются некоторые отношения:

1) структуризации,
2) функциональные,
3) казуальные,
4) семантические.

С помощью первых задаются иерархии информационных единиц, вторые несут процедурную информацию, третьи задают причинно-следственные связи, четвертые соответствуют всем остальным отношениям.

Активность. Исторически сложилось так, что при работе с компьютером активными являются программы, т.е. процедуры, а данные в компьютере оказывались пассивными. Они просто хранились в компьютере и ждали, когда их извлечет оттуда нуждающаяся в них процедура. В интеллектуальных системах актуализации тех или иных действий способствуют знания, имеющиеся в системе. Таким образом, выполнение программы в интеллектуальных системах должно инициироваться текущим состоянием информационной базы. Появление в базе

фактов или описаний событий, установление связей может стать источником активности системы.

Важнейшей проблемой, которую необходимо решить при проектировании интеллектуальной системы, является проблема представления знаний. При проектировании модели представления знаний необходимо учитывать такие факторы, как *однородность* и *простота понимания*. Однородное представление приводит к упрощению механизма управления выводом и упрощению управления знаниями. Представление знаний должно быть понятным экспертам и пользователям системы. В противном случае затрудняются приобретение знаний и их оценка. Однако выполнить это требование весьма сложно.

Рассмотрим типичные модели представления знаний.

ЛОГИЧЕСКИЕ МОДЕЛИ. Описания предметных областей, выполненные в логических языках, называются логическими моделями. Языки логического типа опираются на исчисления, заимствованные из логики.

СЕТЕВЫЕ МОДЕЛИ. В основе моделей этого типа лежит конструкция, называемая семантической сетью. Сетевые модели формально задаются следующим образом:

$$H = <I, C1, C2, \ldots, CN, F>,$$

где I – множество информационных единиц;

$C1, C2, \ldots, CN$ – множество типов связей между информационными единицами;

F – отображение из одного набора связей в другой.

В зависимости от типов связей, используемых в модели, различают классифицирующие сети, функциональные сети и сценарии. В классифицирующих сетях используются отношения структуризации. Функциональные сети характеризуются наличием функциональных отношений. Такие модели называют еще вычислительными моделями. В сценариях используются каузальные отношения, а также отношения типа «средство-результат», «орудие-действие» и т.п. Если в сетевой модели допускаются связи различного типа, то ее обычно называют семантической сетью.

ПРОДУКЦИОННЫЕ МОДЕЛИ. В моделях этого типа в прямой форме представляется информация о процедурах и условиях их применения.

Продукционное правило или *продукция* – это оператор преобразования, представляющий собой выражение следующего вида:

<ситуации> → **<заключение>**.

Левая часть связана с распознаваемой ситуацией и содержит список признаков, которые должны быть выполнены, чтобы было выполнено заключение, содержащееся в правой части.

Возможна и другая трактовка понятия продукции:

<ситуации> → **<действие>**.

Правая часть содержит здесь список действий, которые должны быть выполнены, если выполнен список признаков, описывающих ситуацию.

Продукционная система – это машина, которая воспринимает совокупность известных фактов и строит новые заключения, работая в режиме «распознавание-заключение» (действие).

ФРЕЙМОВЫЕ МОДЕЛИ. Этим моделям присуща жесткая структура информационных единиц, которая называется протофреймом или фреймом-прототипом. В общем случае она выглядит следующим образом:

```
( имя_фрейма:
    имя_слота_1 (значение_слота_1);
    имя_слота_2 (значение_слота_2);
     . . . . . .
    имя_слота_N (значение_слота_N);
).
```

При конкретизации фрейма ему и слотам присваиваются конкретные имена и происходит заполнение слотов. Таким образом из фрейма-прототипа получаются фреймы-экземпляры.

Контрольные вопросы

1. Что изучает дисциплина «Искусственный интеллект»?
2. Чем отличаются алгоритмический и эвристический подходы к решению задач?
3. Какие особенности присущи области применения искусственного интеллекта?
4. В чем заключается принципиальное отличие знаний и данных?
5. На каких формальных теориях построены модели представления знаний?

2 ЛОГИЧЕСКАЯ МОДЕЛЬ

2.1 Логика высказываний

2.1.1 Основы логики высказываний

Употребление термина «логика» в словаре С.И. Ожегова имеет три основных значения:

1) наука о законах и формах мышления;
2) ход рассуждений или умозаключений;
3) разумность, внутренняя закономерность чего-либо.

Таким образом, логику можно рассматривать с различных точек зрения. В данной книге логика используется как формализм для представления знаний. Проблемами, связанными с логическими моделями представления знаний и логическим выводом (не всегда в такой постановке, как в искусственном интеллекте), занимались некоторые из величайших мыслителей, и полученные ими результаты вошли в число наиболее впечатляющих творений естественного интеллекта.

Как самостоятельная научная дисциплина логика сформировалась в силлогистике гениального мыслителя древности Аристотеля. Благодаря работам великих ученых – Авиценны, Канта, Лейбница – логика приобрела математический характер. Но, только начиная с трудов Д. Гильберта, Г. Фреге, Г. Пеано, Б. Рассела и А. Уайтхеда, формальная логика ушла от разговорной речи и выделилась в самостоятельную дисциплину с собственной знаковой системой. Особая роль принадлежит Эрбрану и Робинсону, предложившим автоматический метод доказательства теорем. После того как Р. Ковальски показал, как процесс логического доказательства преобразуется в традиционный процесс вычисления, логика перестала быть сугубо теоретической дисциплиной, став основой для создания языка программирования ПРОЛОГ и породив новое направление в программировании – логическое.

В пособии сначала будет рассмотрена простейшая математическая логика – логика высказываний, или логика нулевого порядка. Здесь основным понятием является высказывание – всякое повествовательное предложение, утверждающее что-либо о чем-либо, – и при этом можно сказать, истинно высказывание или ложно, но ни одно высказывание не может быть одновременно истинным и ложным.

Далее будет рассмотрена более общая система – логика первого порядка. Логика предикатов первого порядка позволяет выразить большее разнообразие утверждений благодаря тому, что в нее добавлены *термы*, *предикаты* и *кванторы*.

В логике высказываний предполагается, что мир может быть описан элементарными предложениями, или высказываниями, и логическими связями между ними. Кроме этого, принято еще два допущения:

1) простому предложению или высказыванию можно приписать истинностное значение;

2) сложные предложения образуются путем видоизменения некоторого предложения с помощью слова «НЕ» (~) или путем связывания простых предложений с помощью слов «И» ($\wedge$); «ИЛИ» ($\vee$); «ЕСЛИ, ТО» ($\rightarrow$); «ТОГДА И ТОЛЬКО ТОГДА, КОГДА» ($\leftrightarrow$). Эти пять слов называют сентенциональными, пропозициональными или логическими связками, каждая из них имеет свое название: « ~ » – отрицание; « $\wedge$ » – конъюнкция; « $\vee$ » – дизъюнкция; « $\rightarrow$ » – импликация; « $\leftrightarrow$ » – эквиваленция.

Таким образом, *высказывание* – это неразлагаемое и неанализируемое повествовательное предложение, которое может быть истинным или ложным, но не тем и другим одновременно. Высказывание, состоящее из одного предложения, называют простым или элементарным.

Существуют два подхода к установлению истинности высказываний: эмпирический и логический. Первый устанавливает истинность высказываний путем выполнения некоторых действий (наблюдений, измерений, эксперимента). Например, пусть есть утверждение «Петя старше Вани», установить истинность данного утверждения можно различными способами: посмотреть их свидетельства о рождении, попытаться определить их возраст визуально. Во втором подходе истинность высказывания устанавливается на основе истинности других высказываний с помощью рассуждений, выявляя связи между высказываниями, входящими в рассуждение. Продолжая рассуждать о возрасте детей… Пусть мы имеем два следующих утверждения, истинность которых установлена: «Петя старше Коли», «Коля старше Вани». Тогда можно сделать вывод, что «Петя старше Вани». Во втором подходе «истина» или «ложь», которая приписывается высказыванию, и есть *истинностное значение* высказывания. Для краткости «истина» обозначается как И, а «ложь» – Л. Высказывания обозначаются заглавными буквами или цепочкой букв. Например, Козьма Прутков считает:

P: «Военные люди защищают отечество»;

Q: «Ветер есть дыхание природы»;

R: «Новые сапоги всегда жмут».

Символы **P**, **Q**, **R** и др., которые используются для обозначения элементарных высказываний, называются *атомарными формулами* или *атомами*.

Примеры сложных высказываний от Козьмы Пруткова: «Чиновник умирает, а его ордена остаются на лице земли»; «Хочешь быть красивым, поступи в гусары». Примем следующие обозначения:

M: чиновник умирает;

L: ордена чиновника остаются на лице земли;

B: хотеть быть красивым;

G: поступать в гусары.

Тогда два последних предложения могут быть записаны в виде формул как $\mathbf{M} \wedge \mathbf{L}$, $\mathbf{B} \rightarrow \mathbf{G}$ соответственно.

2.1.2 Символизация естественного языка средствами логики высказываний

Рассмотрим некоторые аспекты символизации естественного языка и применение логических связок. Операция конъюнкции в логике высказываний и союз «и» в повседневной речи употребляются в одном и том же смысле. Однако в обыденной речи не принято соединять союзом «и» два далеких по смыслу предложения (*ироничное*: «в огороде бузина, а в Киеве дядька»), в то время как в логике высказываний операция конъюнкции соединяет два любых высказывания. Операции конъюнкции соответствуют следующие выражения:

- **A** и **B**;
- не только **A**, но и **B**;
- **B**, хотя и **A**;
- **A**, а также **B**;
- как **A**, так и **B**;
- **A** вместе с **B**.

В повседневной речи союз «или» употребляется в двух различных смыслах: исключающем и неисключающем, а операция дизъюнкции всегда употребляется в неисключающем смысле.

Употребление слов «если …, то …» в повседневной речи существенно отличается от применения в логике высказываний. В предложении «если **A**, то **B**» обыденной речи подразумевается, что **B** логически следует из **A**, в то время как в логике высказываний, не рассматривающей смысла предложений, этого не требуется. Кроме того, ложность предложения **A** в повседневной речи влечет либо ложность **B**, либо потерю смысла всего предложения «если **A**, то **B**». Таким образом, трансляция естественно-языкового предложения в предложение логики высказываний при кажущейся простоте таковой не является. Здесь требуется понять смысл предложения, а затем уже конструировать формулы логики высказываний. Операции импликации ($\mathbf{A} \rightarrow \mathbf{B}$) соответствуют следующие выражения естественного языка:

- **B**, если **A**;
- **A** влечет **B**;
- **A** является причиной **B**;
- **B** является следствием **A**;
- в случае **A** имеет место **B**;
- коль скоро **A**, то **B**;
- **B**, так как **A**;
- **B,** потому что **A**.

Операции эквиваленции (**A** $\leftrightarrow$ **B**) соответствуют следующие выражения естественного языка:

- **A**, если и только если **B**;
- если **A**, то **B,** и обратно;
- **A**, если **B,** и **B**, если **A**;
- **A** эквивалентно **B**;
- **A** равносильно **B**.

2.1.3 Формулы

Правильно построенные формулы, или просто *формулы*, в логике высказываний определяются рекурсивно следующим образом:

- атом есть формула;
- если **G** – формула, то (~**G**) – формула;
- если **G** и **H** – формулы, то (**G** $\vee$ **H**), (**G** $\wedge$ **H**), (**G** $\rightarrow$ **H**) и (**G** $\leftrightarrow$ **H**) – формулы;
- других формул нет.

Логическим связкам приписан следующий убывающий ранг:

$$\leftrightarrow, \rightarrow, \vee, \wedge, \sim .$$

Таким образом, связка с большим рангом имеет большую область действия. Формула **P** $\leftrightarrow$ ~**Q** $\vee$ **R** $\rightarrow$ **S** означает (**P** $\leftrightarrow$ (((~**Q**) $\vee$ **R**) $\rightarrow$ **S**)).

Если **G**, **H** – две формулы, тогда истинность формул (~**G**) , (**G** $\vee$ **H**), (**G** $\wedge$ **H**), (**G** $\rightarrow$ **H**), (**G** $\leftrightarrow$ **H**) определяется по истинностным значениям атомов, входящих в эту формулу, по следующей таблице.

Таблица 2.1 – Истинностная таблица

G	**H**	~**G**	**G** $\wedge$ **H**	**G** $\vee$ **H**	**G** $\rightarrow$ **H**	**G** $\leftrightarrow$ **H**
И	И	Л	И	И	И	И
И	Л	Л	Л	И	Л	Л
Л	И	И	Л	И	И	Л
Л	Л	И	Л	Л	И	И

Интерпретацией формулы является такое приписывание истинностных значений атомам, входящим в формулу, при котором каждо-

му из них приписано либо И, либо Л, но не оба одновременно. Таким образом, если истинностные значения атомов известны, то истинностное значение формулы может быть определено индуктивным способом в соответствии с приведенной выше истинностной таблицей. Если формула содержит n различных атомов, то эта формула имеет 2^n интерпретаций. Ниже приведена истинностная таблица для формул $\mathbf{P} \rightarrow (\mathbf{Q} \rightarrow \mathbf{P})$ и $(\mathbf{P} \rightarrow \mathbf{Q}) \rightarrow \mathbf{P}$ (см. табл. 2.2).

Таблица 2.2

P	**Q**	**P → Q**	**Q → P**	**P → (Q → P)**	**(P → Q) → P**
И	И	И	И	И	И
И	Л	Л	И	И	И
Л	И	И	Л	И	Л
Л	Л	И	И	И	Л

Интерпретацию формулы, содержащей атомы $\mathbf{A_1}$, $\mathbf{A_2}$, …, $\mathbf{A_n}$, удобно представлять в виде

$$I = \{m_1, m_2, \ldots, m_n\},$$

где $\mathbf{m_j}$ есть $\mathbf{A_j}$ или $\mathbf{\sim A_j}$. Если $\mathbf{m_j}$ есть $\mathbf{A_j}$, то атому $\mathbf{A_j}$ присвоено значение **И**, в противном случае **Л**. Например, в интерпретации {**~P, Q**} формула $\mathbf{(P \rightarrow Q) \rightarrow P}$ «ложна», а в интерпретации {**P, ~Q**} эта же формула «истинна».

Формула, истинная при всех возможных интерпретациях, называется *общезначимой формулой*, или *тавтологией*. Формула, ложная при всех возможных интерпретациях, называется *противоречивой* (или *невыполнимой*). Общезначимую и противоречивую формулу будем обозначать «■», «□» соответственно. Формула, которая не является общезначимой или противоречивой, называется *необщезначимой, непротиворечивой или выполнимой* [11].

Общезначимость и противоречивость формулы может быть определена с использованием таблицы истинности, но в силу экспоненциального роста размерности числа интерпретаций с ростом числа входящих в формулу атомов такой метод не всегда является приемлемым. Такое положение привело к необходимости разработки правил преобразования формул. Преобразования выполняются путем замены в преобразуемой формуле некоторой ее части на подформулу, эквивалентную заменяемой. Две формулы *эквивалентны*, если их истинностные значения совпадают при всех интерпретациях. Например, формула $\mathbf{(P \rightarrow Q) \rightarrow P}$ эквивалентна формуле **P**. Для ведения преобразований необходимо иметь минимальный запас эквивалентных формул. Ниже приведены десять законов преобразования, здесь **F**, **G** и **H** являются формулами, символ « = » – это знак эквивалентности.

1. $\mathbf{F} \leftrightarrow \mathbf{G} = (\mathbf{F} \rightarrow \mathbf{G}) \wedge (\mathbf{G} \rightarrow \mathbf{F})$.
2. $\mathbf{F} \rightarrow \mathbf{G} = \sim\mathbf{F} \vee \mathbf{G}$.
3. $\mathbf{F} \vee \mathbf{G} = \mathbf{G} \vee \mathbf{F}$; $\mathbf{F} \wedge \mathbf{G} = \mathbf{G} \wedge \mathbf{F}$.
4. $(\mathbf{F} \vee \mathbf{G}) \vee \mathbf{H} = \mathbf{F} \vee (\mathbf{G} \vee \mathbf{H})$; $(\mathbf{F} \wedge \mathbf{G}) \wedge \mathbf{H} = \mathbf{F} \wedge (\mathbf{G} \wedge \mathbf{H})$.
5. $\mathbf{F} \vee (\mathbf{G} \wedge \mathbf{H}) = (\mathbf{F} \vee \mathbf{G}) \wedge (\mathbf{F} \vee \mathbf{H})$; $\mathbf{F} \wedge (\mathbf{G} \vee \mathbf{H}) = (\mathbf{F} \wedge \mathbf{G}) \vee (\mathbf{F} \wedge \mathbf{H})$.
6. $\mathbf{F} \vee \square = \mathbf{F}$; $\mathbf{F} \wedge \blacksquare = \mathbf{F}$.
7. $\mathbf{F} \vee \mathbf{F} = \mathbf{F}$; $\mathbf{F} \wedge \mathbf{F} = \mathbf{F}$;
8. $\mathbf{F} \vee \blacksquare = \blacksquare$; $\mathbf{F} \wedge \square = \square$;
9. $\mathbf{F} \vee \sim\mathbf{F} = \blacksquare$; $\mathbf{F} \wedge \sim\mathbf{F} = \square$;
10. $\sim(\sim\mathbf{F}) = \mathbf{F}$;
11. $\sim(\mathbf{F} \vee \mathbf{G}) = \sim\mathbf{F} \wedge \sim\mathbf{G}$; $\sim(\mathbf{F} \wedge \mathbf{G}) = \sim\mathbf{F} \vee \sim\mathbf{G}$.

Законы преобразования под номером три называются коммутативными законами; законы под номером четыре – ассоциативными законами; законы под номером пять – дистрибутивными законами; семь – закон идемпотентности; девять – законы дополнения; десять – закон двойного отрицания; одиннадцать – законы де Моргана.

В логике высказываний определены две нормальные формы: дизъюнктивная и конъюнктивная. Формула находится в *дизъюнктивной нормальной форме*, если она имеет следующий вид:

$$\mathbf{F_1} \vee \mathbf{F_2} \vee \ldots \vee \mathbf{F_n},$$

где каждая подформула $\mathbf{F_i}$ – это конъюнкция атомов или отрицания атомов. Формула находится в *конъюнктивной нормальной форме*, если она имеет следующий вид: $\mathbf{F_1} \wedge \mathbf{F_2} \wedge \ldots \wedge \mathbf{F_n}$, где каждая подформула $\mathbf{F_i}$ – это дизъюнкция атомов или отрицания атомов. *Литера* – это атом или отрицание атома. *Дизъюнкт* – это дизъюнкция литер. *Единичный дизъюнкт* – это дизъюнкт, состоящий из одной литеры. Дизъюнкт, не содержащий никаких литер, называется *пустым дизъюнктом*. Формула, находящаяся в конъюнктивной нормальной форме, может быть представлена как множество входящих в форму дизъюнктов. Например, формула $\mathbf{(P \vee Q) \wedge (P \vee R \vee \sim Q) \wedge (\sim Q \vee P) \wedge \sim P}$ может быть представлена как множество $\{\mathbf{P \vee Q}, \mathbf{P \vee R \vee \sim Q}, \mathbf{\sim Q \vee P}, \mathbf{\sim P}\}$.

2.1.4 Вывод в логических моделях нулевого порядка

Задача логики – описать мир в виде логических формул и дать теорию вывода. Практически теория выводов сводится к получению критериев и алгоритмов для решения вопроса о том, можно ли некоторую цепочку рассуждений, основываясь на ее форме, считать правильной. Цепочка рассуждений представляет собой конечную последовательность высказываний, приводимых в обоснование утверждения то-

го, что последнее высказывание в этой последовательности (заключение) может быть выведено из некоторых начальных высказываний (посылок). Это приводит к понятию «логического следствия» [11].

ОПРЕДЕЛЕНИЕ. Пусть даны формулы $\mathbf{F_1}$, $\mathbf{F_2}$, ..., $\mathbf{F_n}$ и формула **G**. Формула **G** есть *логическое следствие* формул $\mathbf{F_1}$, $\mathbf{F_2}$, ..., $\mathbf{F_n}$, если для всякой интерпретации, в которой формула $\mathbf{F_1 \wedge F_2 \wedge ... \wedge F_n}$ истинна, **G** также истинна. Формулы $\mathbf{F_1}$, $\mathbf{F_2}$, ..., $\mathbf{F_n}$ называются посылками, **G** – заключением.

ТЕОРЕМА 1. Пусть даны формулы $\mathbf{F_1}$, $\mathbf{F_2}$, ..., $\mathbf{F_n}$ и формула **G**. Тогда **G** есть логическое следствие формул $\mathbf{F_1}$, $\mathbf{F_2}$, ..., $\mathbf{F_n}$, если формула $\mathbf{((F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow G)}$ общезначима.

ТЕОРЕМА 2. Пусть даны формулы $\mathbf{F_1}$, $\mathbf{F_2}$, ..., $\mathbf{F_n}$ и формула **G**. Тогда **G** есть логическое следствие формул $\mathbf{F_1}$, $\mathbf{F_2}$, ..., $\mathbf{F_n}$, если формула $\mathbf{(F_1 \wedge F_2 \wedge ... \wedge F_n \wedge \sim G)}$ противоречива.

Таким образом, вопрос о том, какие высказывания представляют собой логические следствия других высказываний, сводится к вопросу о том, какие высказывания общезначимы или противоречивы. Это, в свою очередь, дает возможность превратить *ОПРЕДЕЛЕНИЕ* и *ТЕОРЕМЫ 1* и *2* в рабочий аппарат для логического вывода. Ниже приведены восемь способов логического вывода в логике высказываний. В качестве примера рассматриваются следующие формулы: F_1: **P**; F_2: **R**; F_3: $\mathbf{Q \wedge R \rightarrow \sim R}$; G: $\mathbf{\sim Q}$.

Способ 1 – вычисление истинностного значения. Здесь вывод основан на определении логического следствия и на истинностных таблицах (см. табл. 2.3).

Таблица 2.3

P	**R**	**Q**	$\mathbf{Q \wedge R}$	$\mathbf{\sim R}$	$\mathbf{Q \wedge R \rightarrow \sim R}$	$\mathbf{P \wedge R \wedge ((Q \wedge R) \rightarrow \sim R)}$	$\mathbf{\sim Q}$
И	И	И	И	Л	Л	Л	Л
И	И	Л	Л	Л	И	И	И
И	Л	И	Л	И	И	Л	Л
И	Л	Л	Л	И	И	Л	И
Л	И	И	И	Л	Л	Л	Л
Л	И	Л	Л	Л	И	Л	И
Л	Л	И	Л	И	И	Л	Л
Л	Л	Л	Л	И	И	Л	И

Из таблицы 2.3 видно, что есть только одна интерпретация, где формула $\mathbf{P \wedge R \wedge ((Q \wedge R) \rightarrow \sim R)}$ является истинной, в этой же интерпретации формула $\mathbf{\sim Q}$ является истинной, следовательно, формула $\mathbf{\sim Q}$ является логическим следствием формул **P**, **R**, $\mathbf{Q \wedge R \rightarrow \sim R}$.

Способ таблиц истинности 2. Здесь в качестве аппарата для логического вывода может быть использована *ТЕОРЕМА 1* и метод истинностных таблиц.

Таблица 2.4

P	**R**	**Q**	**Q ∧ R**	**~R**	**Q ∧ R → ~R**	**P ∧ R ∧ ((Q ∧ R) → ~R)**	**~ Q**	**F_1∧F_2∧ F_3→ G**
И	И	И	И	Л	Л	Л	Л	И
И	И	Л	Л	Л	И	И	И	И
И	Л	И	Л	И	И	Л	Л	И
И	Л	Л	Л	И	И	Л	И	И
Л	И	И	И	Л	Л	Л	Л	И
Л	И	Л	Л	Л	И	Л	И	И
Л	Л	И	Л	И	И	Л	Л	И
Л	Л	Л	Л	И	И	Л	И	И

Из таблицы 2.4 видно, что формула **P ∧ R ∧ ((Q ∧ R) → ~R) → ~Q** является общезначимой, значит, формула **~Q** является логическим следствием формул **P**, **R**, **Q ∧ R → ~R**.

Способ таблиц истинности 3. В качестве аппарата для логического вывода может быть использована *ТЕОРЕМА 2* и метод истинностных таблиц.

Таблица 2.5

P	**R**	**Q**	**Q ∧ R**	**~R**	**Q ∧ R → ~R**	**P ∧ R ∧ ((Q ∧ R) → ~R)**	**F_1∧F_2∧F_3∧~G**
И	И	И	И	Л	Л	Л	Л
И	И	Л	Л	Л	И	И	Л
И	Л	И	Л	И	И	Л	Л
И	Л	Л	Л	И	И	Л	Л
Л	И	И	И	Л	Л	Л	Л
Л	И	Л	Л	Л	И	Л	Л
Л	Л	И	Л	И	И	Л	Л
Л	Л	Л	Л	И	И	Л	Л

Из таблицы 2.5 видно, что формула **P ∧ R ∧ ((Q ∧ R) → ~R) ∧ ~Q** является противоречивой, значит, формула **~Q** является логическим следствием формул **P**, **R**, **Q ∧ R → ~R**.

Способ 4 – алгебраический. Здесь в качестве аппарата для логического вывода используется ТЕОРЕМА 1, а для доказательства общезначимости формулы – одиннадцать законов эквивалентных преобразований.

P ∧ R ∧ ((Q ∧ R) → ~R) → ~Q = ~(P ∧ R ∧ ((Q ∧ R) → ~R)) ∨ ~Q =
=~(P ∧ R ∧ (~(Q ∧ R) ∨ ~R)) ∨ ~Q = ~P ∨ ~R ∨ (Q ∧ R ∧ R) ∨ ~Q =
=~P ∨ ~R ∨ (Q ∧ R) ∨ ~Q =~P ∨ ~R ∨ (Q ∧ ~Q) ∨ (R ∨ ~Q)=
=~P ∨ ~R ∨ R ∨ ~Q = ~P ∨ ■∨ ~Q = ■.

Способ 5 – также *алгебраический*, только здесь использована *ТЕОРЕМА 2*, а для доказательства противоречивости формулы – десять законов эквивалентных преобразований.

$\mathbf{P \wedge R \wedge ((Q \wedge R) \rightarrow \sim R) \wedge Q = P \wedge R \wedge (\sim(Q \wedge R) \vee \sim R) \wedge Q =}$
$\mathbf{= P \wedge R \wedge (\sim Q \vee \sim R \vee \sim R) \wedge Q = P \wedge R \wedge (\sim Q \vee \sim R) \wedge Q =}$
$\mathbf{= P \wedge R \wedge ((\sim Q \wedge Q) \vee (\sim R \wedge \sim Q)) = P \wedge R \wedge \sim R \wedge \sim Q =}$
$\mathbf{= P \wedge \square \wedge \sim Q = \square}$.

Способ 6 – алгоритм редукции. Этот способ позволяет доказывать общезначимость формул приведением их к абсурду. Этот способ удобен, когда формула содержит много импликаций. Пусть посылка $\mathbf{(P \wedge Q) \rightarrow R}$, а заключение $\mathbf{P \rightarrow (Q \rightarrow R)}$, тогда заключение является логическим следствием посылки, если формула $\mathbf{((P \wedge Q) \rightarrow R) \rightarrow (P \rightarrow (Q \rightarrow R))}$ является общезначимой. Предположим, что в некоторой интерпретации эта формула ложна. Это возможно только тогда, когда формула $\mathbf{(P \wedge Q) \rightarrow R}$ истинна, а формула $\mathbf{P \rightarrow (Q \rightarrow R)}$ ложна. Формула $\mathbf{P \rightarrow (Q \rightarrow R)}$ ложна только тогда, когда $\mathbf{P = И}$, $\mathbf{Q = И}$, $\mathbf{R = Л}$, что противоречит предположению $\mathbf{(P \wedge Q) \rightarrow R = И}$, следовательно, формула $\mathbf{((P \wedge Q) \rightarrow R) \rightarrow (P \rightarrow (Q \rightarrow R))}$ является общезначимой.

Способ 7– алгоритм Девиса и Патнема. Данный алгоритм основан на использовании конъюнктивной нормальной формы, представленной как множество дизъюнктов **S**. Алгоритм состоит из четырех правил.

1. Правило тавтологии. Из множества дизъюнктов **S** вычеркиваются все тавтологичные дизъюнкты.

2. Правило однолитерных дизъюнктов. Если в множестве **S** есть единичный дизъюнкт **C**, то, вычеркнув из множества **S** дизъюнкт **C**, получим множество **S'**. Если **S'** пусто, то **S** выполнимо; иначе, вычеркнув из множества **S'** дизъюнкт **~C**, получим множество **S''**. Множество **S** невыполнимо, когда невыполнимо **S''**. Если **~C** является единичным дизъюнктом, то при его вычеркивании он превращается в противоречивую формулу □.

3. Правило чистых литер. Литера **L** в дизъюнкте из **S** называется чистой, если **~L** не появляется ни в каком дизъюнкте из **S**. Если литера **L** чистая в **S**, то, вычеркнув из множества **S** все дизъюнкты, содержащие **L**, получим множество **S'**. Множество **S** невыполнимо, когда невыполнимо **S'**.

4. Правило расщепления. Если множество дизъюнктов **S** может быть представлено следующим образом:

$$\mathbf{(A_1 \vee L) \wedge \ldots \wedge (A_n \vee L) \wedge (A_1 \vee \sim L) \wedge \ldots \wedge (B_m \vee \sim L) \wedge R},$$

где $\mathbf{A_i}$, $\mathbf{B_j}$, **R** не содержат литер **L** и **~L**, то получим два множества расщепления **S'** = $\mathbf{A_1 \wedge \ldots \wedge A_n \wedge R}$ и **S''** = $\mathbf{B_1 \wedge \ldots \wedge B_m \wedge R}$. Множество **S** невыполнимо, когда невыполнимы множества **S'** и **S''**.

Используя алгоритм Девиса и Патнема, покажем, что формула $\mathbf{(P \vee Q) \wedge (P \vee R) \wedge (\sim Q \vee \sim R) \wedge \sim P \wedge U}$ невыполнима.

S = { P ∨ Q , P ∨ R, ~Q ∨ ~R, ~P, U };

{ Q, R, ~Q ∨ ~R, U } правило 2, единичный дизъюнкт **C** = **~P**;

{ R, ~R, U } правило 2, единичный дизъюнкт **C** = **Q**;

{□, U } правило 2, единичный дизъюнкт **C** = **R**.

Так как множество содержит невыполнимый дизъюнкт, то оно невыполнимо.

Способ 8 – метод резолюций. Данный метод является обобщением правила однолитерных дизъюнктов Девиса и Патнема и также основан на использовании конъюнктивной нормальной формы, представленной как множество дизъюнктов **S**. Обобщение правила связано с тем, что оно применяется к любой паре дизъюнктов, не обязательно единичных. Правило это называется правилом резолюций и формулируется следующим образом:

🕮 ***Если в дизъюнкте C_1 существует литера L, а в дизъюнкте C_2 существует литера ~L, то, вычеркнув литеры L и ~L из C_1 и C_2, соответственно, построим дизъюнкцию оставшихся дизъюнктов, которая называется резольвентой дизъюнктов C_1 и C_2.***

Рассмотрим, например, дизъюнкты $\mathbf{C_1}$: **P ∨ Q**, C_2: **~Q ∨ ~R ∨ U**, резольвентой дизъюнктов $\mathbf{C_1}$ и $\mathbf{C_2}$ будет следующий дизъюнкт: **P ∨ ~R ∨ U**.

🕮 ***<u>Основное свойство резольвенты</u>. Любая резольвента дизъюнктов C_1 и C_2 является логическим следствием дизъюнктов C_1 и C_2. Для невыполнимого множества дизъюнктов применением метода резолюций можно получить пустой дизъюнкт □.***

Метод резолюций основан на проверке того, содержит ли исходное множество дизъюнктов пустой дизъюнкт: если множество содержит пустой дизъюнкт, то это множество невыполнимо, в противном случае проверяется, можно ли получить пустой дизъюнкт из исходного множества. Такой процесс проверки называется выводом.

Выводом из конечного множества дизъюнктов **S** называется конечная последовательность $\mathbf{C_1}$, $\mathbf{C_2}$, …, $\mathbf{C_n}$ дизъюнктов**,** всякий элемент $\mathbf{C_i}$ которой или принадлежит множеству **S**, или является резольвентой дизъюнктов, предшествующих данному элементу $\mathbf{C_i}$**.** Вывод пустого дизъюнкта □ из **S** называется опровержением или доказательством невыполнимости **S**.

Используя метод резолюций, проведем вывод и покажем, что формула $\mathbf{(P \vee Q) \wedge (P \vee R) \wedge (\sim Q \vee \sim R) \wedge \sim P \wedge U}$ невыполнима. Запи-

шем множество дизъюнктов **S** = **{ P ∨ Q , P ∨ R, ~Q ∨ ~R, ~P, U }** следующим образом:

1) **P ∨ Q;**
2) **P ∨ R;**
3) **~Q ∨ ~R;**
4) **~P;**
5) **U;**
6) из п.п. 2, 3 получим резольвенту **P ∨ ~Q;**
7) из п.п. 1, 6 получим резольвенту **P;**
8) из п.п. 4, 7 получим резольвенту □.

Так как □ есть логическое следствие множества дизъюнктов **S**, то **S** невыполнимо.

Доказать невыполнимость конечного множества дизъюнктов **S** можно с помощью следующего алгоритма.

Шаг 1. Если множество **S** содержит пустой дизъюнкт, то останов с сообщением «Множество невыполнимо».

Шаг 2. Если возможно найти резольвенту, то вычислить резольвенту **R**, иначе останов с сообщением «Множество выполнимо».

Шаг 3. **S**:= **S** ∪ **{ R }.** Перейти на шаг 1.

Приведенный алгоритм – недетерминированный, на шаге 2 возможно вычисление различных резольвент, некоторые резольвенты могут оказаться ненужными и вычисляться несколько раз, а алгоритм без надлежащих проверок может зациклиться. Построение оптимального компьютерного алгоритма выходит за рамки данного учебного пособия.

🕮 ***Если возможно описать задачу в терминах логики высказываний, то, применив любой из указанных восьми способов вывода, можно, доказав противоречивость или общезначимость формулы, решить поставленную задачу.***

2.2 Логика предикатов первого порядка

2.2.1 Основы логики предикатов

Класс задач, решаемых с использованием логики высказываний, очень ограничен. Например, из посылок следующего классического примера – «Все люди смертны» и «Сократ – человек» – интуитивно следует заключение «Сократ смертен».

Однако в рамках логики высказываний решить эту задачу не удается. Объясняется это тем, что утверждение в логике высказываний – это неделимый объект (атом), а приведенный пример требует анализа внутренней структуры предложения. Логика предикатов первого по-

рядка позволяет выразить большее разнообразие утверждений благодаря тому, что в нее добавлены *термы*, *предикаты* и *кванторы*.

Если проанализировать приведенные выше три утверждения, то можно обнаружить, что рассуждения здесь ведутся на некоторой предметной области (множестве людей). «Сократ» является объектом этой предметной области. Кроме того, в первом утверждении есть неявное указание на принадлежность к этой предметной области («если некто принадлежит к множеству людей, значит, он смертен»). Такое неконкретизированное указание объекта предметной области соответствует понятию «*переменная*», а явное указание «Сократ» – понятию «*константа*». Принадлежность объекта к предметной области можно задать в виде «логической функции» или предиката. Например, предикат **ЧЕЛОВЕК(x)** указывает на то, что если **x** является человеком, то высказывание **ЧЕЛОВЕК(x)** является истинным, и, соответственно, предикат **СМЕРТЕН(x)** указывает на то, что **x** смертен. Здесь формы записи **ЧЕЛОВЕК** и **СМЕРТЕН** называются предикатными символами. Выражение «все люди» служит примером квантификации, такая запись символически может быть представлена как $\forall$ и называется квантором всеобщности (общности). Запись **($\forall$x)** читается как «для всех **x**», «для всякого **x**», «для каждого **x**». Тогда приведенные выше три высказывания можно формально записать следующим образом:

($\forall$x)(ЧЕЛОВЕК(x) $\rightarrow$ СМЕРТЕН(x))
ЧЕЛОВЕК(Сократ**)**
СМЕРТЕН(Сократ**)**

Кроме квантора всеобщности, в логике предикатов используется квантор существования, символически записываемый в виде **($\exists$x)**, что означает «существует **x**», «для некоторых **x**», «по крайней мере, для одного **x**».

Константы и переменные образуют более общее понятие – *терм*, который определяется следующим образом:

- константа – это терм;
- переменная – это терм;
- если f – n-местный функциональный символ и $t_1, t_2, \ldots, t_n$ – термы, то $f(t_1, t_2, \ldots, t_n)$ – терм;
- других термов нет.

Синонимом для сложного терма вида $f(t_1, t_2, \ldots, t_n)$ будет «функция». Функция есть отображение списка констант в константу. Предикат – это тоже отображение списка констант, но не в константу, а в элемент множества {И, Л}.

📖 ***Основными символьными конструкциями логики предикатов являются константы, переменные, термы, предикатные символы, логические связки, кванторы.***

После определения терма можно дать понятие атома и формулы в логике предикатов. Если P – n-местный предикатный символ и $t_1, t_2, \ldots, t_n$ – термы, то $P(t_1, t_2, \ldots, t_n)$ – *атом*. Понятие *формулы* рекурсивно определяется следующим образом:

- атом – это формула;
- если **G** и **H** – формулы, то (~**G**), (**G** $\wedge$ **H**), (**G** $\vee$ **H**), (**G** $\rightarrow$ **H**) и (**G** $\leftrightarrow$ **H**) – формулы.
- если **G** – формула и **x** – переменная, то **(∀x)G**, **(∃x)G** – формулы.
- других формул нет.

Здесь, как и в логике высказываний, круглые скобки могут быть опущены в соответствии с принятыми соглашениями о ранге логических связок, считая ранг кванторов наименьшим.

Примем следующие соглашения по использованию мнемонических имен, которые будут действовать по умолчанию:

- константы обозначаются строчными буквами a, b, c, d, e, возможно, с индексами;
- переменные – это обычно строчные буквы x, y, z, u, v, w, возможно, с индексами;
- функциональные символы представляются буквами f, g, h или словами из строчных букв;
- предикатные символы обозначаются прописными буквами P, Q, R, S или словами из прописных букв.

2.2.2 Символизация естественного языка средствами логики предикатов

Рассмотрим несколько примеров *формализации предложений естественного языка*. Пусть **R(x)** обозначает «**x** есть рациональное число», а **Q(x)** – «**x** есть действительное число», тогда предложение «каждое рациональное число есть число действительное» можно переформулировать в такое: «для любого x, если x есть рациональное число, то x есть действительное число» – и символически представить в следующем виде: **(∀x)(R(x)** $\rightarrow$ **Q(x))**.

Смысл этого предложения сводится к тому, что множество рациональных чисел является подмножеством множества действительных чисел, что записывается как **R** $\subseteq$ **Q**. Такая запись является типичной для высказываний «всякое нечто есть то-то».

Теперь рассмотрим утверждение «некоторые действительные числа являются рациональными». Формула, правильно отражающая это предложение, будет иметь вид: **(∃x)(Q(x) ∧ R(x)).**

Это означает, что пересечение двух множеств действительных и рациональных чисел не является пустым $\mathbf{R} \cap \mathbf{Q} \neq \varnothing$. Типичная ошибка при формализации такого типа предложений заключается в том, что, исходя из правильности формализации предложений типа «всякое нечто есть то-то» в виде **(∀x)(R(x) → Q(x))**, делается ошибочное предположение в правильности формализации предложения типа «некоторое нечто есть то-то» формулой **(∃x)(Q(x) → R(x))**.

И еще одно предложение ошибочно – «не каждое действительное число есть число рациональное», что может быть прочтено как «не верно, что каждое действительное число является рациональным», формальная запись которого выглядит так: **~((∀x)(Q(x) → R(x)))**.

Переформулировав это предложение еще и как «существуют числа, которые являются действительными, но не являются рациональными», можно это же предложение представить и в виде такой формулы: **(∃x)(Q(x) ∧ ~R(x))**.

Таким образом, имеются две формулы, отражающие одно и то же предложение естественного языка, однако в дальнейшем станет ясно, что эти формулы являются эквивалентными.

📖 ***Не существует простых синтаксических правил перевода предложений естественного языка в язык формул логики предикатов. Для перевода необходимо установить смысл предложения, а потом транслировать этот смысл в язык логики предикатов.***

2.2.3 Интерпретация

Формулы логики предикатов могут быть интерпретированы, т.е. им может быть приписано истинностное значение. Однако наличие термов требует присвоения и им значения из предметной области. Таким образом, *интерпретация* формулы *F* логики предикатов состоит из непустой предметной области *D* и задания значений из данной предметной области всем константам, переменным и функциям, встречающимся в формуле *F*; каждому предикату, определенному на *D*, приписывается либо И, либо Л. Правила интерпретации для неквантифицированных и квантифицированных формул будут следующие:

- если истинностные значения формул **G** и **H** заданы, то истинностные значения формул (**~G**), (**G ∧ H**), (**G ∨ H**), (**G → H**) и (**G ↔ H**) определяются по таблицам истинности;

- формула **(∀x)G** получит значение **И**, если значение **И** получит формула **G** для каждого **x** из области *D*, в противном случае формула **G** получит значение **Л**;
- формула **(∃x)G** получит значение **И**, если формула **G** получит значение **И** хотя бы для одного **x** из области *D*, в противном случае формула **G** получит значение **Л**.

Рассмотрим следующий пример: пусть дана формула

(∀x) (∃y)(P(x, a) → Q(f(y))),

в следующей интерпретации: область *D* = {1,2}, константа **a** = 1 – функция **f** принимает следующие значения:

f(1)	f(2)
2	1

оценка для предикатов следующая:

P(1, 1)	P(1, 2)	P(2, 1)	P(2, 2)	Q(1)	Q(2)
И	Л	И	Л	И	Л

Пусть **x** = 1, **y** = 1, определим истинностное значение формулы **P(x, a) → Q(f(y))** при данных значениях **x** и **y**.

P(x, a) → Q(f(y)) = **P(1, a) → Q(f(1))** = **P(1, 1) → Q(2) = И → Л = Л**.

Положим **y** = 2.

P(x, a) → Q(f(y)) = **P(1, a) → Q(f(2))** = **P(1, 1) → Q(1) = И → И = И**, т.е. для **x** = 1 существует такой **y** (а именно **y** = 2), что формула **P(1, a) → Q(f(y))** принимает значение **И**.

Пусть теперь **x** = 2, **y** = 1, определим истинностное значение формулы **P(x, a) → Q(f(y))** при данных значениях **x** и **y**.

P(x, a) → Q(f(y)) = **P(2, a) → Q(f(1))** = **P(2, 1) → Q(2) = И → Л = Л**.

Положим **y** = 2.

P(x, a) → Q(f(y)) = **P(2, a) → Q(f(2))** = **P(2, 1) → Q(1) = И → И = И**.

Таким образом, и для **x** = 2 существует такой **y** (а именно **y** = 2), что формула **P(2, a) → Q(f(y))** принимает значение **И**. Так как для всех **x** из области *D* существует такое значение **y**, что формула **P(x, a) → Q(f(y))** истинна, то формула **(∀x) (∃y)(P(x, a) → Q(f(y)))** истинна в указанной интерпретации.

Формулы логики предикатов, также как и формулы логики высказываний, делятся на три класса: *общезначимые* – истинные во всех своих интерпретациях, *противоречивые* (*невыполнимые*) – ложные во всех своих интерпретациях, и *непротиворечивые* (*выполнимые*) – истинные только в отдельных своих интерпретациях [11].

2.2.4 Нормальные формы

Понятие логического следствия формул, определенное для формул логики высказываний, справедливо и в логике предикатов; справедливы здесь и отношения между общезначимостью, противоречивостью и логическим следствием, заданные в виде *ТЕОРЕМ 1* и *2*. В общем случае язык логики высказываний является подмножеством языка логики предикатов первого порядка. Формула логики высказываний может рассматриваться как формула логики предикатов без кванторов, функций и переменных.

📖 ***Формула логики предикатов имеет в общем случае бесконечное число областей интерпретации или предметных областей и, как следствие, бесконечное число интерпретаций, а значит, невозможно доказать противоречивость или общезначимость формулы перебором всех ее интерпретаций***.

Цель дальнейшего изложения – определить процедуру проверки противоречивости формул логики предикатов. Для решения этой задачи введены еще две нормальные формы: *предваренная и сколемовская стандартная*, помимо двух ранее определенных: конъюнктивной и дизъюнктивной. Введение нормальных форм позволит упростить алгоритмы поиска противоречивости формул логики предикатов.

Предваренной (префиксной) нормальной формой называется формула, состоящая из префикса и матрицы, здесь префикс – это конечная последовательность кванторных комплексов, а матрица – это формула, не содержащая кванторных комплексов, т.е. формула имеет следующий вид:

$$(Q_1x_1)\,(Q_2x_2)\,\ldots\,(Q_nx_n)\,M,$$

где Q_i есть $\forall$ или $\exists$ для i = 1, 2, …, n.

Рассмотрим метод преобразования формулы логики предикатов в предваренную нормальную форму. Одиннадцать законов преобразования были определены ранее при рассмотрении логики высказываний (см. п. 2.1.3), эти законы действуют и в логике предикатов. Ниже рассмотрим дополнительные пары эквивалентных формул, содержащих кванторы (пусть **F** и **H** – это формулы, содержащие переменную **x**, **G** есть формула, которая не содержит переменной **x**, а **Q** – это $\forall$ или $\exists$):

12) $(Qx)F(x) \wedge G = (Qx)(F(x) \wedge G)$; $(Qx)F(x) \vee G = (Qx)(F(x) \vee G)$;
13) $\sim((\forall x)\,F\,(x)) = (\exists x)(\sim F(x))$; $\sim((\exists x)F(x)) = (\forall x)(\sim F(x))$.
14) $(\forall x)F(x) \wedge (\forall x)H(x) = (\forall x)(F\,(x) \wedge H(x))$;
$(\exists x)F(x) \vee (\exists x)H(x) = (\exists x)(F(x) \vee H(x))$.

Переменная в формулах логики предикатов может быть переименована. На использовании этого приема основаны обобщающие законы (п. 14) следующие правила преобразования:

15. $(Q_1x)F(x) \vee (Q_2x)H(x) = (Q_1x)F(x) \vee (Q_2z)H(z) = (Q_1x)\,(Q_2z)\,(F(x) \vee \vee H(z))$,
$(Q_3x)F(x) \wedge (Q_4x)H(x) = (Q_3x)F(x) \wedge (Q_4z)H(z) = (Q_3x)\,(Q_4z)\,(F(x) \wedge \vee H(z))$.

Любая формула логики предикатов допускает эквивалентную предваренную нормальную форму. Эскиз процедуры преобразования приведен ниже.

Шаг 1. Исключить логические связки эквиваленции и импликации.

Шаг 2. Используя законы двойного отрицания, де Моргана и законы под номером двенадцать, пронести знак отрицания внутрь формулы.

Шаг 3. Если необходимо, то переименовать переменные.

Шаг 4. Используя остальные законы, вынести кванторы в начало формулы.

Введение кванторов в формулы логики предикатов позволило увеличить выразительную мощность, одновременно усложнив язык данного формализма. Упрощение языка возможно за счет приемлемого уменьшения выразительной мощности. Идея упрощения состоит в том, чтобы, сохранив противоречивость формулы, в ней исключить кванторы существования путем использования сколемовских функций. Тогда произвольной формуле **F** логики предикатов сопоставляется некоторая формула $\mathbf{S_F}$, причем, если формула **F** невыполнима, то и формула $\mathbf{S_F}$ также невыполнима. Такая связь между формулами **F** и $\mathbf{S_F}$ строго слабее, чем логическая эквивалентность, однако именно такое сопоставление позволяет провести доказательство невыполнимости формулы более эффективно.

Сколемовской стандартной формой называется формула, находящаяся в предваренной форме, у которой матрица приведена к конъюнктивной нормальной форме. Получить сколемовскую стандартную форму из произвольной формулы логики предикатов можно, используя следующую процедуру.

Шаг 1. Привести данную формулу к предваренной нормальной форме.

Шаг 2. Привести матрицу к конъюнктивной нормальной форме.

Шаг 3. Если квантора существования в префиксе нет, то полученная формула и есть сколемовская стандартная форма, в противном

случае выбирается самый левый кванторный комплекс существования в префиксе **(∃x)**.

Шаг 4. Если никакой квантор всеобщности не стоит в префиксе левее выбранного квантора существования, то, выбрав новую константу **a**, отличную от других констант, входящих в матрицу, заменим все **x** в матрице на константу **a**, вычеркнем кванторный комплекс существования **(∃x)** из префикса. Перейти на шаг 3.

Шаг 5. Если в префиксе левее выбранного кванторного комплекса существования **(∃x)** стоят кванторные комплексы всеобщности $(\forall y_1)\ \ldots(\forall y_m)$, т.е. выбранный квантор существования находится в области действия кванторов всеобщности, что означает наличие зависимости **x** от $y_1, \ldots y_m$, тогда, выбрав новый *m*-местный функциональный символ **f**, отличный от других функциональных символов, входящих в матрицу, заменим все **x** в матрице на функцию $f(y_1, \ldots y_m)$, вычеркнем кванторный комплекс существования **(∃x)** из префикса. Перейти на шаг 3.

Константы и функции, используемые для замены переменных квантора существования, называются *сколемовскими функциями*.

Рассмотрим пример, пусть необходимо получить сколемовскую стандартную форму формулы

$$(\forall x)(\exists y)P(x, y) \to (\forall z)(\exists v)((\forall w)Q(z, w) \to (\exists w)R(z, v, w)) =$$
$$=\sim((\forall x)(\exists y)P(x, y)) \vee (\forall z)(\exists v)((\forall w)Q(z, w) \to (\exists w)R(z, v, w)) =$$
$$=\sim((\forall x)(\exists y)P(x, y)) \vee (\forall z)(\exists v)(\sim((\forall w)Q(z, w)) \vee (\exists w)R(z, v, w)) =$$
$$=((\exists x)(\forall y)\sim P(x, y)) \vee (\forall z)(\exists v)(\sim((\forall w)Q(z, w)) \vee (\exists w)R(z, v, w)) =$$
$$=(\exists x)(\forall y)(\sim P(x, y)) \vee (\forall z)(\exists v)((\exists w)(\sim Q(z, w)) \vee (\exists w)R(z, v, w)) =$$
$$=(\exists x)(\forall y)(\sim P(x, y)) \vee (\forall z)(\exists v)(\exists w)(\sim Q(z, w) \vee R(z, v, w)) =$$
$$=(\exists x)(\forall y)(\forall z)(\exists v)(\exists w)\ (\sim P(x, y) \vee \sim Q(z, w) \vee R(z, v, w)) =$$
$$\{x = a, v = f(y, z), w = g(y, z)\} =$$
$$=(\forall y)(\forall z)(\sim P(a, y) \vee \sim Q(z, g(y, z)) \vee R(z, f(y, z), g(y, z))).$$

Сколемовская стандартная форма – это предваренная форма, префикс которой содержит только кванторы всеобщности. Поэтому удобно префикс опустить, считая, что каждая переменная в матрице управляется квантором всеобщности, а так как конъюнктивная форма – это конъюнкция дизъюнктов, то сколемовскую стандартную форму можно представить в виде множества дизъюнктов.

📖 ***Сколемовская стандартная форма – это множество дизъюнктов.***

Например, стандартная форма

$$(\forall y)(\forall z)(\sim P(a, y) \vee \sim Q(z, g(y, z)) \vee R(z, f(y, z), g(y, z)))$$

может быть представлена множеством

$$S = \{\sim P(a, y) \vee \sim Q(z, g(y, z)) \vee R(z, f(y, z), g(y, z))\}.$$

2.2.5 Выводы в логических моделях первого порядка

Формула логики предикатов противоречива в том случае, когда противоречиво множество дизъюнктов, представляющих стандартную форму этой формулы, а противоречиво множество дизъюнктов тогда, когда оно ложно при всех интерпретациях во всех предметных областях.

📖 ***Черч и Тьюринг: Не существует общего универсального метода, позволяющего определить общезначимость или противоречивость формулы логики предикатов первого порядка.***

📖 ***Однако существуют алгоритмы, подтверждающие общезначимость или противоречивость формулы, если формула действительно общезначима или противоречива. Для необщезначимых или непротиворечивых формул алгоритмы в общем случае свою работу не заканчивают.***

Частные, но интересные с точки зрения определения процедуры поиска доказательства противоречивости формулы, результаты были получены Эрбраном. Так как рассмотрение всех возможных интерпретаций формулы логики предикатов в общем случае невозможно, Эрбраном была найдена специальная универсальная область интерпретации. Стандартная форма формулы противоречива тогда и только тогда, когда форма ложна при всех интерпретациях в этой области. Называют эту область *эрбрановским универсумом*, и определяется она для множества дизъюнктов **S** следующим образом:

- множество констант нулевого уровня $\mathbf{H_0}$ состоит из констант, встречающихся в **S**; если **S** не содержит констант, тогда $\mathbf{H_0}$ содержит одну произвольно выбранную константу, допустим, $\mathbf{H_0} = \{\mathbf{c}\}$;
- множество констант **i**-го уровня (i = 1, 2, …, ∞) определяется как объединение констант уровня $\mathbf{i-1}$ и множества всех термов $\mathbf{f(t_1, t_2, \ldots, t_n)}$, где **f** – все функциональные символы, встречающиеся в **S**, а $\mathbf{t_1, t_2, \ldots, t_n}$ – константы уровня $\mathbf{i-1}$;
- множество $\mathbf{H_\infty}$ есть эрбрановский универсум множества дизъюнктов **S.**

Элементы эрбрановского универсума – это абстрактные объекты, не имеющие конкретной интерпретации. Если во множестве **S** отсутствуют функции, то эрбрановский универсум всегда конечен и состоит из множества констант. Если же множество **S** содержит хотя бы одну функцию, то универсум всегда бесконечен.

Рассмотрим несколько примеров. Пусть множество дизъюнктов

$\mathbf{S = \{P(x, y) \vee \sim Q(x, z, u), \sim P(u, y) \vee R(y) \vee Q(x, y, u), S(x)\}}$, тогда $\mathbf{H_\infty = \{c\}}$.

Для множество дизъюнктов

S = {**Q(a, g(y)), P(x, y)**}, универсум $\mathbf{H_\infty}$ **=**{**a, g(a), g(g(a)), g(g(g(a))), …**}.

Если множество дизъюнктов

S = {~P(a, y) ∨ ~Q(z, g(y, z)) ∨ R(z, f(y, z), g(y, z))}, то

$\mathbf{H_\infty}$ **={a, g(a, a), f(a, a), g(a, g(a, a)), g(g(a, a), a), g(g(a, a), g(a, a)), g(a, f(a, a)), g(f(a, a), a), g(f(a, a), f(a, a)), f(a, g(a, a)), f(g(a, a), a), f(g(a, a), g(a, a)), …}.**

Эрбрановской интерпретацией, или *H-интерпретацией* для множества дизъюнктов **S,** называется интерпретация, удовлетворяющая следующим условиям:

- предметной областью является эрбрановский универсум;
- интерпретация отображает все константы из **S** в соответствующую эрбрановскую константу;
- если **f** – **n**-местный функциональный символ и $\mathbf{h_1, h_2, …, h_n}$ – константы эрбрановского универсума, то в эрбрановской интерпретации через **f** обозначается функция, отображающая ($\mathbf{h_1, h_2, …, h_n}$) в $\mathbf{f(h_1, h_2, …, h_n)}$.

В общем случае эрбрановских интерпретаций на множестве **S** может быть бесконечно много, так как интерпретации предикатов и функций могут быть выбраны произвольно. Пусть множество дизъюнктов

S = {**P(x, y) ∨ ~Q(x, z, u)**, ~**P(u, y) ∨ R(y) ∨ Q(x, y, u), S(x)**},

эрбрановский универсум для этого множества дизъюнктов – $\mathbf{H_\infty}$ **=**{**c**}. Некоторые интерпретации приведены ниже:

$\mathbf{I_1}$ = {**P(c, c), Q(c, c, c), R(c), S(c)**}, $\mathbf{I_2}$ = {~**P(c, c), Q(c, c, c), R(c), S(c)**},
$\mathbf{I_3}$ = {**P(c, c), ~Q(c, c, c), R(c), S(c)**}, $\mathbf{I_4}$ = {**P(c, c), Q(c, c, c), ~R(c), S(c)**}.

Так как здесь имеются четыре атома **P(c, c), Q(c, c, c), R(c), S(c)**, то всего существуют $2^4 = 16$ эрбрановских интерпретаций.

Для множества дизъюнктов **S** = {**Q(a, g(y)), P(x, y)**} эрбрановский универсум бесконечен $\mathbf{H_\infty}$ **=**{**a, g(a), g(g(a)), g(g(g(a))), …**}, и, соответственно, эрбрановских интерпретаций будет бесконечное множество, четыре из них приведены ниже:

$\mathbf{I_1}$ **= {P(a, a), Q(a, a), P(a, g(a)), Q(a, g(a)), P(g(a), a), Q(g(a), a), P(g(a), g(a)), Q(g(a), g(a)), …},**

$\mathbf{I_2}$ **= {~P(a, a), Q(a, a), ~P(a, g(a)), Q(a, g(a)), ~P(g(a), a), Q(g(a), a), ~P(g(a), g(a)), …},**

$\mathbf{I_3}$ **= {P(a, a), ~Q(a, a), P(a, g(a)), ~Q(a, g(a)), P(g(a), a), ~Q(g(a), a), P(g(a), g(a)), …},**

$\mathbf{I_4}$ **= {~P(a, a), ~Q(a, a), ~P(a, g(a)), ~Q(a, g(a)), ~P(g(a), a), ~Q(g(a), a), ~P(g(a), g(a)), …}.**

Доказано, что множество дизъюнктов **S** невыполнимо тогда и только тогда, когда оно ложно при всех своих эрбрановских интерпретациях.

📖 ***Для проверки выполнимости множества дизъюнктов необходимо рассматривать только эрбрановские интерпретации***.

Выражение – это терм, множество термов, множество атомов, множество дизъюнктов. Если выражение не содержит переменных, то оно называется *основным*.

Основной пример дизъюнкта **C** множества дизъюнктов **S** есть дизъюнкт, полученный заменой переменных в **C** на константы эрбрановского универсума **S**. Пусть множество дизъюнктов **S** = {**Q(a, g(y)), P(x, y)**}, дизъюнкт **C** = **P(x, y)**, эрбрановский универсум $\mathbf{H_\infty}$ **=**{**a, g(a), g(g(a)), g(g(g(a))), …**}, тогда основной пример **C' = P(g(a), a)**.

Основной пример выполняется в данной интерпретации тогда и только тогда, когда в этом основном примере существует основная литера, которая есть и в данной интерпретации. Пусть множество дизъюнктов **S** = {**Q(a, g(y)), P(x, y)**}, дизъюнкт **C** = **P(x, y)**, основной пример **C' = P(g(a), a)**, шесть интерпретаций:

$\mathbf{I_1}$ = **{P(a, a), Q(a, a), P(a, g(a)), Q(a, g(a)), P(g(a), a), Q(g(a), a), P(g(a), g(a)), Q(g(a), g(a)), …}**,

$\mathbf{I_2}$ = **{~P(a, a), Q(a, a), ~P(a, g(a)), Q(a, g(a)), ~P(g(a), a), Q(g(a), a), ~P(g(a), g(a)), …}**,

$\mathbf{I_3}$ = **{P(a, a), ~Q(a, a), P(a, g(a)), ~Q(a, g(a)), P(g(a), a), ~Q(g(a), a), P(g(a), g(a)), …}**,

$\mathbf{I_4}$ = **{~P(a, a), ~Q(a, a), ~P(a, g(a)), ~Q(a, g(a)), ~P(g(a), a), ~Q(g(a), a), ~P(g(a), g(a)), …}**,

$\mathbf{I_5}$ = **{~P(a, a), Q(a, a), P(a, g(a)), Q(a, g(a)), P(g(a), a), Q(g(a), a), P(g(a), g(a)), …},**

$\mathbf{I_6}$ = **{~P(a, a), ~Q(a, a), P(a, g(a)), Q(a, g(a)), P(g(a), a), Q(g(a), a), P(g(a), g(a)), …}**,

тогда основной пример **C'** выполняется в интерпретации $\mathbf{I_1}$**,** $\mathbf{I_3}$**,** $\mathbf{I_5}$**,** $\mathbf{I_6}$ и опровергается в $\mathbf{I_2}$**,** $\mathbf{I_4}$.

Дизъюнкт выполняется в данной интерпретации тогда и только тогда, когда каждый основной пример выполняется в данной интерпретации.

Дизъюнкт опровергается в данной интерпретации тогда и только тогда, когда существует хотя бы один основной пример данного дизъюнкта, который не выполняется в данной интерпретации.

Вернемся к предыдущему примеру, дизъюнкт **C** = **P(x, y)** выполняется в интерпретации $\mathbf{I_1}$**,** $\mathbf{I_3}$ и опровергается в $\mathbf{I_2}$**,** $\mathbf{I_4}$**,** $\mathbf{I_5}$**,** $\mathbf{I_6}$.

Множество дизъюнктов невыполнимо тогда и только тогда, когда для каждой интерпретации существует хотя бы один основной пример дизъюнкта, невыполнимый в данной интерпретации.

Пусть **S** = {**P(x)** ∨ **~Q(x)**, **~P(x)**, **Q(x)**}, на данном множестве дизъюнктов существуют четыре интерпретации:

$\mathbf{I_1}$ = {**P(c), Q(c)**},

$\mathbf{I_2}$ = {**~P(c), Q(c)**},

$\mathbf{I_3}$ = {**P(c), ~Q(c)**},

$\mathbf{I_4}$ = {**~P(c), ~Q(c)**}.

В интерпретации $\mathbf{I_1}$ опровергается дизъюнкт **~P(x)**; дизъюнкт **P(x)** ∨ **~Q(x)** опровергается в интерпретации $\mathbf{I_2}$; интерпретация $\mathbf{I_3}$ опровергает дизъюнкт **Q(x)**; этот же дизъюнкт **Q(x)** опровергается в интерпретации $\mathbf{I_4}$. Рассматриваемое множество дизъюнктов невыполнимо, потому что оно опровергается во всех интерпретациях.

📖 ***Теорема Эрбрана. Множество дизъюнктов невыполнимо тогда и только тогда, когда существует конечное невыполнимое множество основных примеров дизъюнктов указанного множества.***

Рассмотрим предыдущий пример, **S** = {**P(x)** ∨ **~Q(x)**, **~P(x)**, **Q(x)**}, приведенное множество дизъюнктов невыполнимо потому, что существует следующее невыполнимое множество основных примеров:

S' = {**P(c)** ∨ **~Q(c)**, **~P(c)**, **Q(c)**}.

На основе теоремы Эрбрана может быть создана компьютерная процедура, генерирующая множества основных примеров и проверяющая их невыполнимость. Одним из первых, кто осуществил это, был П. Гилмор. Он в 1960 году написал компьютерную программу, которая порождала множества основных примеров, полученных заменой переменных во множестве дизъюнктов на константы эрбрановского универсума. Так как множество основных примеров – это конъюнкция дизъюнктов, не содержащих переменных (по сути, это высказывания), то можно воспользоваться любым из восьми способов проверки противоречивости данного множества при условии, что множество основных примеров конечно.

Процедуры поиска опровержения, основанные на теореме Эрбрана, имеют один существенный недостаток – они требуют генерации множеств $\mathbf{S'_1}$, $\mathbf{S'_2}$, …, $\mathbf{S'_n}$, … основных примеров рассматриваемого множества дизъюнктов **S**. Очень часто размерность множеств основных примеров растет экспоненциально, по этой причине такие процедуры не имеют практического применения на современных компьютерах. И только с появлением метода резолюций, разработанного Дж. Робинсоном, были найдены эффективные компьютерные процедуры доказательства невыполнимости множества дизъюнктов. Идея метода Робинсона состоит в том, чтобы работать напрямую с дизъюнктами, входящими в рассматриваемое множество, не порождая ос-

новных примеров. Реализация метода резолюций требует применения операции унификации. Но прежде чем объяснить эту операцию, дадим определение понятия подстановки.

Подстановка – это конечное множество вида $\{v_1 = t_1, v_2 = t_2, \ldots, v_n = t_n\}$, где v_j – это переменная, а t_j – это терм, отличный от v_j, причем все v_j отличны друг от друга.

Унифицировать два или более выражений – значит найти такую подстановку, которая делает выражения одинаковыми (тождественными). Такая подстановка называется *унификатором*.

Рассмотрим два выражения **P(x, a, f(a, g(y)))** и **P(h(v), z, f(z, u))**. Они не тождественны, однако, если применить к ним операцию унификации с подстановкой $\{x = h(v), z = a, u = g(y)\}$, то получим два тождественных выражения **P(h(v), a, f(a, g(y)))**.

В логическом исчислении удобно работать с дизъюнктами, не содержащими повторяющихся литер. Устранение повторения в результирующем дизъюнкте выполняют с помощью операции склейки. Если две или более литер с одинаковым знаком в дизъюнкте **C** имеют общий унификатор $\{v_1 = t_1, v_2 = t_2, \ldots, v_n = t_n\}$, то дизъюнкт, полученный из **C** заменой одновременно всех вхождений переменных v_i на термы t_i, называется *склейкой* дизъюнкта **C**.

Применение операции унификации и склейки позволяет использовать метод резолюций в логике предикатов первого порядка.

Пусть $\mathbf{C_1}$ и $\mathbf{C_2}$ – дизъюнкты, не имеющие общих переменных. Если в дизъюнкте $\mathbf{C_1}$ существует литера $\mathbf{L_1}$, а в дизъюнкте $\mathbf{C_2}$ существует литера $\mathbf{\sim L_2}$, и литеры $\mathbf{L_1}$ и $\mathbf{\sim L_2}$ унифицируемы, то, вычеркнув литеры $\mathbf{L_1}$ и $\mathbf{\sim L_2}$ из $\mathbf{C_1}$ и $\mathbf{C_2}$ соответственно, построим дизъюнкцию оставшихся дизъюнктов, которая называется *бинарной резольвентой* дизъюнктов $\mathbf{C_1}$ и $\mathbf{C_2}$.

Рассмотрим пример: пусть $\mathbf{C_1}$ **= P(x, g(b)) ∨ ~Q(x)** и $\mathbf{C_2}$ **= ~P(a, g(x)) ∨ R(z)**. Сначала переименуем переменную **x** в первом дизъюнкте: $\mathbf{C_1}$ **= P(y, g(b)) ∨ ~Q(y)**. Выберем в дизъюнкте $\mathbf{C_1}$ литеру $\mathbf{L_1}$**=P(y, g(b))** и литеру $\mathbf{L_2}$**= ~P(a, g(x))** в $\mathbf{C_2}$. Литеры $\mathbf{L_1}$ и $\mathbf{\sim L_2}$ имеют общий унификатор $\{y = a, x = b\}$, тогда бинарная резольвента имеет следующий вид: **~Q(a))) ∨ R(z)**.

Резольвентой дизъюнктов $\mathbf{C_1}$ и $\mathbf{C_2}$ является одна из следующих резольвент:

бинарная резольвента $\mathbf{C_1}$ и $\mathbf{C_2}$;

бинарная резольвента $\mathbf{C_1}$ и склейки $\mathbf{C_2}$;

бинарная резольвента $\mathbf{C_2}$ и склейки $\mathbf{C_1}$;

бинарная резольвента склейки $\mathbf{C_1}$ и склейки $\mathbf{C_2}$.

Пусть $\mathbf{C_1 = \sim P(x, y) \vee \sim P(f(z), z) \vee \sim Q(x, y)}$ и $\mathbf{C_2 = P(f(g(a)), g(a)) \vee R(a)}$. Склейка $\mathbf{C_1}$ имеет следующий вид: $\mathbf{\sim P(f(z), z) \vee \sim Q(f(z), z).}$ Бинарная резольвента склейки $\mathbf{C_1}$ и $\mathbf{C_2}$ равна $\mathbf{\sim Q(f(g(a)), g(a)) \vee R(a)}$.

Контрольные вопросы

1. Какое минимальное число пропозициональных связок необходимо для эквивалентного представления формул логики высказываний?
2. Даны следующие высказывания:

A: Иванов купил компьютер.

B: Петров успешно сдал экзамен.

C: Сидоров уехал в другой город.

Переведите на естественный язык следующие формулы логики высказываний:

$\sim A \rightarrow B$; $(\sim A \wedge B) \rightarrow C$;

$\sim(\sim A \rightarrow B)$; $\sim(B \wedge C) \rightarrow \sim A$;

$\sim(\sim A \rightarrow B) \rightarrow C$; $\sim(\sim B \wedge \sim C) \rightarrow A$.

3. Перед началом соревнований трое болельщиков высказали предположения относительно будущих победителей.

1-й болельщик: «Спартак» будет первым, «Динамо» займет третье место.

2-й болельщик: победителем будет «Динамо», «Торпедо» займет третье место.

3-й болельщик: первое место займет «Торпедо», «Динамо» будет вторым.

По окончании соревнований выяснилось, что каждый из них был прав лишь в одном из своих предположений. Опишите распределение мест, если каждое место было занято только одним клубом.

4. Доказать общезначимость следующих формул:

$\mathbf{(\exists x)\,(\exists y)P(x, y) \leftrightarrow (\exists y)\,(\exists x)\,P(x, y)}$;

$\mathbf{(\exists x)\,(\forall y)P(x, y) \rightarrow (\forall y)\,(\exists x)\,P(x, y)}$.

3 ПРОДУКЦИОННАЯ МОДЕЛЬ

3.1 Представление

Впервые метод продукций был предложен Эмилем Постом в 1936 г. в контексте уточнения понятия «алгоритм». Данный вычислительный формализм был основан на правилах замены строк символов. В 1963 г. этот метод был использован Ноамом Хомским для определения языков и формальных грамматик. А с начала семидесятых годов метод продукций находит широкое применение для представления знаний в экспертных системах и в системах, основанных на знаниях.

Продукционное правило, или *продукция*, – это оператор преобразования, представляющий собой выражение следующего вида:

<ситуация> → **<заключение>**.

Левая часть связана с распознаваемой ситуацией и содержит список признаков, которые должны быть выполнены, чтобы было выполнено заключение, содержащееся в правой части.

Возможна и другая трактовка понятия продукции:

<ситуация> → **<действие>**.

Здесь правая часть содержит список действий, которые должны быть выполнены, если выполнен список признаков, описывающих ситуацию.

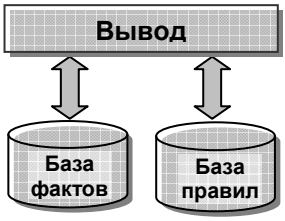


Рис. 3.1

Продукционная система – это машина, которая воспринимает совокупность известных фактов и строит новые заключения, работая в режиме распознавание – заключение (действие). Продукционная система состоит из трех основных компонентов (рис. 3.1). Первый из них – это набор правил, используемый как база знаний, иногда его еще называют *базой правил*. Второй компонент – это *база фактов* или *рабочая память* – память для временного хранения, в которой хранятся предпосылки, касающиеся конкретных задач предметной области, и результаты выводов, получаемые на их основании. Третий компонент реализует *механизм логического вывода*, обрабатывающий правила в соответствии с содержанием рабочей памяти; другое название этого компонента – машина логического вывода.

Способ получения вывода представляет собой либо правило *модус поненс*:

посылка	p,
правило	p → q,
заключение	q;

либо *модус толленс*:

посылка ~q
правило $p \rightarrow q$,
заключение ~p.

Рабочая память (РП) в начале работы продукционной системы содержит исходные данные для поставленной задачи и пополняется фактами, установленными в процессе работы системы. Правила продукций применяются к содержимому рабочей памяти. Как было указано выше, правило состоит из двух частей: условной и заключительной. Условная часть может состоять из одного или нескольких условий. Если условная часть правила удовлетворяет содержимому рабочей памяти, то правило может быть применено, что приведет к изменению рабочей памяти. Машина логического вывода выполняет следующие действия:

1) определяет, какое именно правило следует выбрать;
2) выполняет выбранное правило и дополняет рабочую память;
3) прекращает вычисления, если выполнено одно из 2-х условий:
   - ❑ выполнено условие останова;
   - ❑ невозможно применить ни одного правила из базы правил.

Указанные выше действия положены в основу алгоритма работы системы продукций. Рассмотрим пример продукционной системы с простым набором правил:

$C \wedge D \rightarrow A$, $B \rightarrow A$, $E \rightarrow C$, $F \rightarrow C$, $G \wedge H \rightarrow D$, $J \wedge K \wedge L \rightarrow B$.

В правилах из показанного примера ситуации описываются буквами, например, правило: $C \wedge D \rightarrow A$ означает следующее:

ЕСЛИ наблюдается как ситуация C, так и ситуация D,

ТО имеет место ситуация A.

Пусть в рабочей памяти имеются следующие факты: J, K, E, F, G, H. Тогда работу продукционной системы можно представить как последовательное применение правил и переписывание содержимого рабочей памяти (см. рис. 3.2):

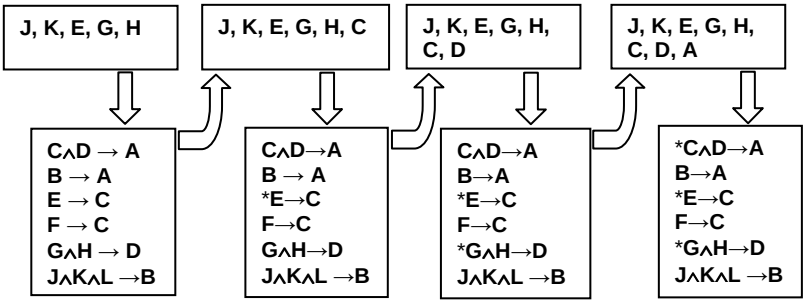


Рис. 3.2

Машина логического вывода сопоставляет образцы из условной части правил с образцами, хранимыми в рабочей памяти. Если все образцы присутствуют в рабочей памяти, тогда правило выполняется, в противном случае – отклоняется. Будем предполагать также, что из всех возможных кандидатов выполняется первое (самое верхнее) правило. После применения правило выбывает из числа кандидатов (выгорает), и его вторичное применение невозможно. В нашем случае сначала будет применено правило E→C, а рабочая память дополнится фактом C, затем отработает правило G∧H→D, в рабочую память будет помещен факт D, и наконец, после применения правила C∧D→A рабочая память пополнится фактом A. После чего система остановится, так как правил для выполнения больше не будет.

Такой способ получения новых фактов называется *прямым выводом* или выводом от данных к цели потому, что поиск новых сведений здесь происходит в направлении стрелки, разделяющей левые и правые части правил. Вывод в нашем случае проведен в три этапа. Цепочка логического вывода, порождаемая в приведенном примере, показана на рис. 3.3.

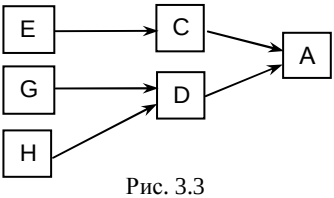


Рис. 3.3

Предположим, что наша продукционная система использовалась с целью определения того, возможно ли получение факта A из фактов J, K, E, F, G, H. Тогда можно считать, что система отработала успешно, так как позволила установить факт A, не сделав лишней работы. Однако, если базу правил дополнить еще двумя правилами E∧J→M и G→M и в качестве исходных данных оставить J, K, E, F, G, H, то в зависимости от последовательности применения правил общий останов случится через три этапа или через пять. Здесь могут отработать еще и два вновь введенных правила, которые отдалят получение целевого факта A. Этот несложный пример показывает, что выбор применяемого правила оказывает прямое влияние на эффективность вывода. В реальной системе с несколькими тысячами правил это становится крайне сложной проблемой.

Увеличение числа шагов логического вывода произошло за счет вывода новых фактов, хотя и справедливых, но никак не связанных с получением целевого факта A. Поэтому, если речь идет о выводе единственного конкретного факта, то в этом случае прямой вывод может

оказаться неэффективным. В такой ситуации, с точки зрения затрат, более целесообразным может оказаться использование процедуры *обратного вывода*. При использовании такого способа вывода система начинает работу с формулировки того, что требуется доказать (например, предполагая, что ситуация A присутствует и выполняет только те правила, которые имеют отношение к доказательству предположения). Ниже показано, каким образом работает процедура обратного вывода при использовании правил из предыдущего примера.

Шаг 1. В систему поступает указание о необходимости вывода A. Сначала осуществляется проверка наличия A в базе фактов и после отрицательного ответа поиск правил, заканчивающихся на A, то есть имеющих A справа от стрелки. Система находит два правила $C \wedge D \rightarrow A$, $B \rightarrow A$ и принимает решение о том, что для подтверждения A необходимо установить наличие или C и D, или B.

Шаг 2. Система пытается установить C в результате проверки сначала базы фактов и последующего обнаружения правил, приводящих к C. Обнаружено два таких правила: $E \rightarrow C$ и $F \rightarrow C$. Система приходит к решению о том, что для получения вывода о присутствии C ей необходимо установить наличие или E, или F.

Шаг 3. Система обнаруживает E в базе фактов, подтвердив возможность вывода C.

Шаг 4. Система пытается установить D сначала в базе фактов, потом в результате поиска правил, приводящих к D. Обнаружено одно такое правило $G \wedge H \rightarrow D$. Система приходит к решению о том, что для получения вывода о присутствии D ей необходимо установить наличие G и H.

Шаг 5. Система обнаруживает G и H в базе фактов, подтвердив возможность вывода D.

Шаг 6. Система выполняет правило $E \rightarrow C$ и устанавливает факт C.

Шаг 7. Система выполняет правило $G \wedge H \rightarrow D$ и устанавливает факт D.

Шаг 8. Система выполняет правило $C \wedge D \rightarrow A$ и устанавливает факт A.

Сформированная в данном случае цепочка логического вывода не отличается от образованной при выполнении процедуры прямого вывода. Различие двух подходов связано со способом, с помощью которого осуществляется поиск фактов и правил.

Два позже введенных правила $E \wedge J \rightarrow M$ и $G \rightarrow M$ в обратном выводе не участвуют, однако здесь так же, как и при прямом выводе, возникает проблема выбора. На шаге 1 необходимо определить, какое правило сначала подтверждать: $C \wedge D \rightarrow A$ или $B \rightarrow A$, а на шаге 2 необходимо выбрать между $E \rightarrow C$ или $F \rightarrow C$.

Если на каждом этапе логического вывода существует множество применяемых правил, то это множество носит название *конфликтного набора*, а выбор одного из них называется разрешением конфликта. Чтобы повысить эффективность продукционной системы, необходимо решить проблему управления последовательностью применения правил или управления выводом.

Рассмотренные выше правила представляют собой отношение вывода, установленное между содержимым рабочей памяти, ссылка на которое осуществляется из условной части (УЧ), и содержимым, указываемым в заключительной части (ЗЧ). Визуально такое отношение можно представить в виде графа с древовидной структурой (рис. 3.4).

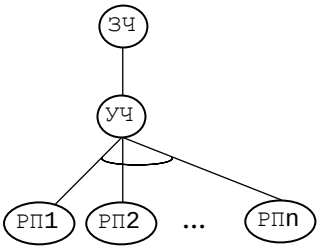


Рис. 3.4

Здесь РП1, РП2, …, РПn – данные, хранящиеся в рабочей памяти, к которым обращается данное правило.

Если существует множество правил, из которых выводится одно и то же заключение, то, выполнив операцию ИЛИ над всеми заключениями, получаемыми с помощью этих правил, можно показать отношение между результатами отдельного вывода и данными, на основании которых делается вывод (рис. 3.5).

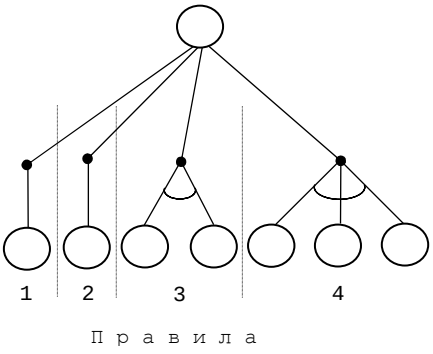


Рис. 3.5

Следовательно, если в такой форме представить отношение между всеми правилами продукционной системы и содержимым РП, то систему можно представить в виде одного графа типа И/ИЛИ. В самых нижних узлах этого графа будут располагаться основные системные данные, а в самом верхнем узле – заключения, выводимые системой. В итоге вывод, получаемый с помощью продукционной системы, можно

представить как совокупность серии правил, поддерживающих отдельное заключение, и данных, на основании которых делается вывод.

Ниже рассмотрен пример определенной ранее продукционной системы с шестью правилами при выводе факта *A* (рис. 3.6).

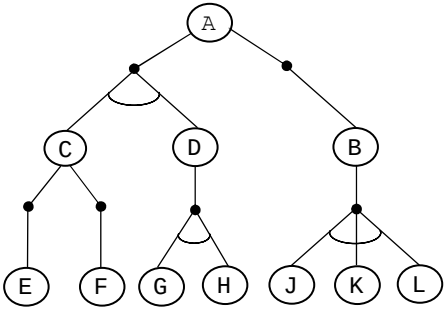


Рис. 3.6

С помощью такого графа обратный вывод можно представить как проблему поиска пути на данном графе. Другими словами, для подтверждения одной цели из всех связей ИЛИ, определенных по отношению к узлам графа, соответствующим этой цели, выбирается одна, и делается попытка подтвердить все узлы, являющиеся предусловиями этой цели. В случае неудачи выбирается следующая связка ИЛИ и повторяется аналогичная процедура. Если обнаружится, что хотя бы одна из связок ИЛИ позволяет вывести последнюю цель, то доказательство этой цели считается успешным, в противном случае – неуспешным. Выбор одной из связок ИЛИ есть не что иное, как выбор одного из правил, таким образом, мы возвращаемся к проблеме разрешения конфликтов. Кроме того, определение последовательности оценки связок И, раскрывающих связки ИЛИ, соответствует последовательности оценки УЧ-правил.

3.2 Вывод

Как уже отмечалось нами, разрешение конфликтов - важная проблема, касающаяся эффективности функционирования продукционной системы. Следовательно, чтобы повысить эту эффективность, необходимо решить проблему управления последовательностью применения правил или управления выводом.

При прямом выводе в системе циклически выполняются следующие действия [6]:

- поиск данных в рабочей памяти,

- генерация конфликтного набора,
- разрешение конфликта,
- применение правил.

Для управления выводом здесь используются два подхода:

1) установка ограничений на генерацию конфликтного набора,
2) определение алгоритма разрешения конфликта.

Первый подход. Здесь в зависимости от содержания правил либо используется метод, по которому поиск условной части правил определенной категории не осуществляется, либо используется метод, по которому правила предварительно разбиваются на отдельные категории, и в определенных ситуациях исследуется возможность применения правил, принадлежащих только к некоторой категории.

Первый метод реализуется с использованием метаправил, т.е. правил, в условных частях которых содержится условие, касающееся содержания правил и содержимого рабочей памяти.

Реализация второго метода предполагает, что правила предварительно группируются по атрибутам. Для каждой группы указывается условие, касающееся содержимого рабочей памяти, и исследуется возможность применения правила только из этой группы; либо группа указывается с помощью заключительной части правила и допускается или запрещается применение правил этой группы.

Если при прямом выводе все заключения, которые можно определить, обнаруживаются, то проблема разрешения конфликтов решается сравнительно просто. Здесь можно использовать простой способ, согласно которому правила применяются в соответствии с последовательностью их определения. В случаях, когда задано условие останова, выполнение этого условия можно ускорить путем управления выводом, например, углублением этапа вывода насколько это возможно. Этого можно достичь посредством выполнения вывода по приоритету глубины, что фактически соответствует попытке приоритетного применения правила со ссылкой на последнюю информацию, которой была дополнена рабочая память.

На рисунке 3.6 показано, что последним было выполнено *правило 1*. При сравнении *правил-кандидатов 2 и 3* предпочтение отдается *правилу 2*, как наиболее быстро приближающемуся к искомому факту. При данном способе путь логического вывода минимален.

Проблема ограничения конфликтного набора и выбора алгоритма разрешения конфликтов характерна и для систем с обратным выводом.

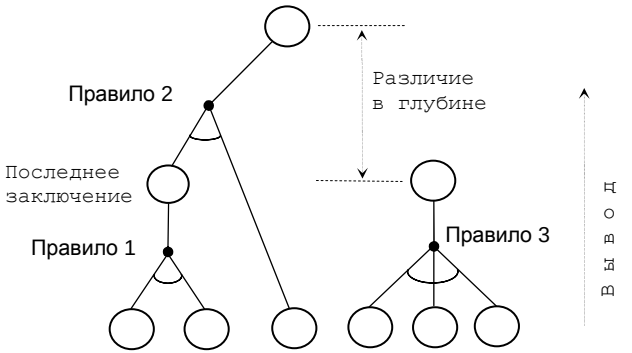


Рис. 3.6

Разбиение правил на группы ограничивает размеры конфликтных наборов, порожденных одновременно, и уменьшает потребность в поиске содержимого РП. Эти методы широко используются для повышения эффективности системы и реализации стратегии вывода.

Типичным примером такого подхода является модель доски объявлений (blackboard model). Впервые этот подход был применен при разработке системы распознавания речи HEARSAY-II. Суть его заключается в определении для одной рабочей памяти группы правил, называемых источниками знаний (ИЗ). Вся рабочая память разбивается на уровни, например, уровень слогов и уровень слов. Каждый источник знаний определяется как совокупность правил, выполняющих специальные функции между уровнями рабочей памяти (см. рис. 3.7). Например, выполняющих вывод между уровнем отдельных слов и уровнем слогов. Таким образом, можно построить систему, делающую выводы путем запуска этих источников знаний в соответствии с ситуацией. Причем рабочая память в ней будет являться «доской объявлений», совместно используемой источниками знаний, и в то же время представлять собой средство для извлечения информации.

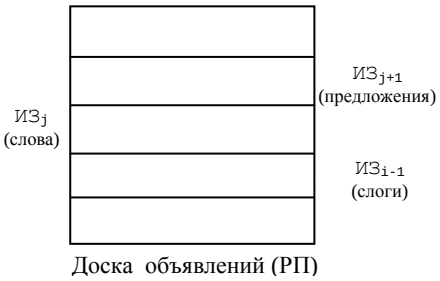


Рис. 3.7

3.3 Система OPS5

OPS5 – представитель семейства OPS (средства разработки систем продукций) – разработан в Университете Карнеги-Меллона [8]. OPS5 является языком программирования, ориентированным на правила. Правила управляются данными и работают на единственной базе данных, называемой рабочей памятью. Данные в OPS5 являются либо векторами, либо объектами со связанными с ними парами атрибут-значение. Например, утверждение, что система управления по борьбе с утечкой химических веществ должна локализовать источник утечки, прежде чем пытаться оградить ее, может быть представлено как вектор:

(ПОРЯДОК_ВЫПОЛНЕНИЯ ЛОКАЛИЗАЦИЯ_ИСТОЧНИКА
ОГРАЖДЕНИЕ_УТЕЧКИ).

Утверждение о том, что серная кислота является бесцветной и принадлежит к классу кислот, можно представить как элемент атрибут-значение:

(ВЕЩЕСТВО ^ИМЯ H2SO4 ^ЦВЕТ БЕСЦВЕТНОЕ ^КЛАСС КИСЛОТА).

Символ «^» является оператором, который отделяет атрибуты от их значений. Таким образом, элемент рабочей памяти описывается так:

<элемент рабочей памяти> ::= < вектор-элемент > | < аз-элемент>
< вектор-элемент > ::= ({< значение >}$^{+}$)
< аз-элемент> ::= (<объект> {^ < атрибут > < значение >}$^{+}$).

Элементы в РП могут динамически изменяться по размеру в процессе работы программы. Векторы могут получать или терять значения, а *аз*-элементы могут получать или терять пары атрибут-значение.

Правила имеют вид <антецедент> $\rightarrow$ <консеквент>, где антецедент является частичным описанием данных, а консеквент – одно или более действий, которые должны быть выполнены, если антецеденты соответствуют содержимому рабочей памяти. Управление в OPS5 осуществляется циклом «опознал-выполнил». В цикле ищутся правила с удовлетворившимися антецедентами, из них выбирается одно правило (разрешение конфликта) и выполняются действия.

Упрощенный язык OPS5, в котором не рассматриваются векторные элементы и другие особенности OPS5 (например, использование связок И/ИЛИ), в антецедентах можно описать следующим образом:

<правило> ::= (P < имя_правила > <антецедент> $\rightarrow$ <консеквент>)
<антецедент> ::= {< условие >}$^{+}$
<условие> ::= < образец > | - < образец >
< образец > ::= (<объект> {^ < атрибут > < значение >}$^{+}$).
<консеквент> ::= {< действие >}$^{+}$

< действие > ::= (СОЗДАТЬ <объект> {^ < атрибут > < значение >}$^{+}$) |
(МОДИФИЦИРОВАТЬ <образец-число>
{^ < атрибут > < значение >}$^{+}$) |
(УДАЛИТЬ <образец-число>) |
(ЗАПИСАТЬ {< значение >}$^{+}$).

Значениями в антецеденте и консеквенте могут быть числа, символы или переменные. В OPS5 переменной называется символ, который заключен в ломаные скобки, например, <X>, <ПЕРВЫЙ>. Образцы в антецеденте представляют собой частичные описания элементов данных. Интерпретатор определяет, сопоставим ли образец с элементом данных, сравнивая подэлементы образца с соответствующими подэлементами данных. Каждый подэлемент условия должен быть сопоставим с соответствующим подэлементом данных по следующим правилам:

1) символ константы или числа сопоставим только с равной константой;

2) переменная сопоставима с любым символом или числом, однако, если переменная в антецеденте встречается несколько раз, то все вхождения этой переменной должны быть сопоставимы с одним и тем же значением.

Говорят, что переменная связана с элементом, с которым она сопоставима. Например, образец

(ВЕЩЕСТВО ^КЛАСС КИСЛОТА ^ИМЯ <ВЕЩ>)

будет сопоставим с элементом данных

(ВЕЩЕСТВО ^ИМЯ H2SO4 ^ЦВЕТ БЕСЦВЕТНОЕ ^КЛАСС КИСЛОТА)

и связывает переменную <ВЕЩ> с элементом (H2SO4). Другой пример, образец (ПОРЯДОК-ВЫПОЛНЕНИЯ <ПЕРВЫЙ>) будет сопоставим с элементом данных

(ПОРЯДОК_ВЫПОЛНЕНИЯ ЛОКАЛИЗАЦИЯ_ИСТОЧНИКА
ОГРАЖДЕНИЕ_УТЕЧКИ),

связывая переменную <ПЕРВЫЙ> с элементом (ЛОКАЛИЗАЦИЯ-ИСТОЧНИКА).

OPS5 предусматривает ряд операторов для изменения значений подэлемента условия. Следующие три оператора особенно важны: префиксный оператор ≠ и два вида скобок – фигурные ({,}) и двойные ломаные (<<, >>). Первый оператор – это оператор неравенства. Пара «≠ *значение*» сопоставима с любыми данными, лишь бы они не были сопоставимы с элементом *значение*. Фигурные скобки указывают, что все заключенные в них значения должны быть сопоставимы с одним и тем же подэлементом рабочей памяти. Таким образом, запись ^ЗНАЧЕНИЕ {<X> ≠ NIL} будет сопоставима с любым элементом ЗНАЧЕНИЕ, который не равен NIL, и связывает с ним переменную <X>.

Другой тип скобок, двойные ломаные, указывает, что подэлемент рабочей памяти сопоставим с любым из подэлементов, заключенных в этот вид скобок.

Как следует из описания символьной структуры <условие>, перед образцами в антецеденте может стоять оператор «-». Антецедент удовлетворяется, если все образцы, перед которыми не стоит минус, сопоставляются с элементами данных и никакой из образцов, перед которым стоит минус, не сопоставляется с элементами данных. Таким образом, следующий антецедент:

(ЦЕЛЬ ^СОСТОЯНИЕ АКТИВНЫЙ ^ИМЯ УКАЗАТЬ_КОНТРМЕРЫ)
(ИСТОЧНИК ^ТИП ПОСТОЯННЫЙ_РЕЗЕРВУАР ^МЕСТОПОЛОЖЕНИЕ <АТ>) - (КОНТРМЕРЫ ^МЕСТОПОЛОЖЕНИЕ <АТ> ^ВИД КАНАВА)

может быть описан на естественном языке следующим образом:

«Если имеется активная цель, указать контрмеры из числа уже существующих, и имеется источник, являющийся постоянным резервуаром, и не существует контрмеры типа канавы, находящейся в том же месте …».

Наиболее важными типами действий являются: [СОЗДАТЬ], [УДАЛИТЬ], [МОДИФИЦИРОВАТЬ] и [ЗАПИСАТЬ].

[СОЗДАТЬ] создает и добавляет к рабочей памяти еще один новый элемент. Например, запись вида

(СОЗДАТЬ ВЕЩЕСТВО ^ИМЯ H2SO4 ^ЦВЕТ БЕСЦВЕТНОЕ ^КЛАСС КИСЛОТА)

добавит новый элемент

(ВЕЩЕСТВО ^ИМЯ H2SO4 ^ЦВЕТ БЕСЦВЕТНОЕ ^КЛАСС КИСЛОТА).

Действие [УДАЛИТЬ] уничтожает в рабочей памяти один или несколько элементов. Например, (УДАЛИТЬ 1) уничтожит элемент, с которым был успешно сопоставлен первый образец антецедента.

[МОДИФИЦИРОВАТЬ] изменяет один или несколько подэлементов существующего элемента. Действие

(МОДИФИЦИРОВАТЬ 1 ^СОСТОЯНИЕ ОЖИДАНИЕ)

изменит

(ЦЕЛЬ ^СОСТОЯНИЕ АКТИВНЫЙ ^ТРЕБУЕТСЯ ОБРАБОТАТЬ_ВРЕДНОЕ_ВЕЩЕСТВО)

на

(ЦЕЛЬ ^СОСТОЯНИЕ ОЖИДАНИЕ ^ТРЕБУЕТСЯ ОБРАБОТАТЬ_ВРЕДНОЕ_ВЕЩЕСТВО).

[ЗАПИСАТЬ] выводит информацию на терминал пользователя. Действие

(ЗАПИСАТЬ (CRLF) ВВЕДИТЕ НАЗВАНИЕ ВЫТЕКАЮЩЕГО ВЕЩЕСТВА:)

ВВЕДИТЕ НАЗВАНИЕ ВЫТЕКАЮЩЕГО ВЕЩЕСТВА: осуществить переход на новую строку (CRLF) и затем печатает:

ВВЕДИТЕ НАЗВАНИЕ ВЫТЕКАЮЩЕГО ВЕЩЕСТВА:

Рассмотрим правило координации деятельности системы, принимающее решение о наилучшем порядке выполнения своих подзадач.

(Р КООРДИНИРОВАТЬ-А (ЦЕЛЬ ^ИМЯ КООРДИНИРОВАТЬ ^СОСТОЯНИЕ АКТИВНЫЙ) - (ПОРЯДОК-ЗАДАЧ) →
(СОЗДАТЬ ЦЕЛЬ ^ИМЯ УПОРЯДОЧИТЬ_ЗАДАЧИ ^СОСТОЯНИЕ АКТИВНЫЙ)
(МОДИФИЦИРОВАТЬ 1 ^СОСТОЯНИЕ ОЖИДАНИЕ)

«Если имеется цель, скоординировать задачи системы, которые являются активными, И предпочтительного порядка нет, ТО создать подцель, определить предпочтительный порядок, сделав ее состояние активным, И модифицировать цель «координировать», изменив ее на состояние ожидание».

Интерпретатор OPS5 исполняет систему продукций, делая следующие операции:

1) шаг *сопоставление*. Определить, какие правила имеют удовлетворительные антецеденты;

2) шаг *разрешение конфликта*. Выбрать одно правило с удовлетворенным антецедентом. Если правил с удовлетворенным антецедентом нет, остановить выполнение;

3) шаг *действие*. Выполнить действия выбранного правила;

4) шаг *возврат*. Перейти к шагу один.

Эту последовательность действий можно рассматривать, как общую схему структуры управления, которую пользователь наполняет по своему усмотрению, поскольку в OPS5 сама система продукций определяет, какие будут использованы стратегии управления и решения задач.

Управление в системе продукций может быть реализовано при помощи явных целей, т.е. элементов в рабочей памяти, определяющих требуемые состояния. Каждое правило в системе содержит образец, сопоставимый с целью определенного типа, и поэтому каждое правило срабатывает только тогда, когда в рабочей памяти находятся цели данного типа. Например, в приведенном выше примере первый образец в правиле КООРДИНИРОВАТЬ-А сопоставляется с целью типа КООРДИНИРОВАТЬ И, таким образом, это правило выполняется только тогда, когда активная цель этого типа находится в рабочей памяти. Система продукций способна управлять своей собственной работой, помещая цели в рабочую память или удаляя их в соответствии с выбранной стратегией. Например, правило КООРДИНИРОВАТЬ-А, помещая цель УПОРЯДОЧИТЬ_ЗАДАЧИ в рабочую память и делая ее активной, заставляет систему обслужить новую задачу – задачу генерации предпочтительной последовательности для последующей обработки.

OPS5 – это специализированная система с прямыми выводами, эффективность выполнения которых увеличена благодаря применению алгоритма согласования RETE при генерации конфликтного набора. По этому алгоритму каждый раз при дополнении рабочей памяти новым образцом проверяется правило, в котором этот образец использу-

ется, и если образец удовлетворяет условной части правила, то он запоминается именно в этом качестве, и только если добавление образца удовлетворяет всем условиям, данное правило включается в конфликтный набор. В OPS5 применяется метод разрешения конфликта LEX: предпочтение здесь отдается правилам со ссылкой на самый последний сгенерированный элемент рабочей памяти. Среди этих элементов выбирается правило с наибольшим числом условий в условной части. В последних версиях OPS алгоритм разрешения конфликтов частично может определять пользователь.

Контрольные вопросы

1. Дайте определение понятию «продукционное правило».
2. Определите способы вывода в продукционных системах.
3. Разработайте продукционные правила для системы локализации неисправностей телевизора (автомобиля, компьютера, другой сложной системы).
4. Определите понятие «конфликтный набор», покажите, как будет решена проблема конфликта в разработанной Вами системе.
5. Представьте разработанные Вами правила на языке OPS5.

4 ФРЕЙМОВАЯ МОДЕЛЬ

4.1 Представление

Фреймовая модель, или модель представления знаний, основанная на фреймовой теории М. Минского, представляет собой систематизированную в виде единой теории психологическую модель памяти человека и его сознания. Важным моментом в этой теории является понятие фрейма – структуры данных для представления некоторого концептуального объекта. Информация, относящаяся к этому фрейму, содержится в слотах. Все фреймы взаимосвязаны и образуют единую сеть фреймов. Однако четкого определения связи между фреймами и слотами может и не быть. Рассмотрим пример фреймовой системы, описывающей аудиторию 426 (рис. 4.1).

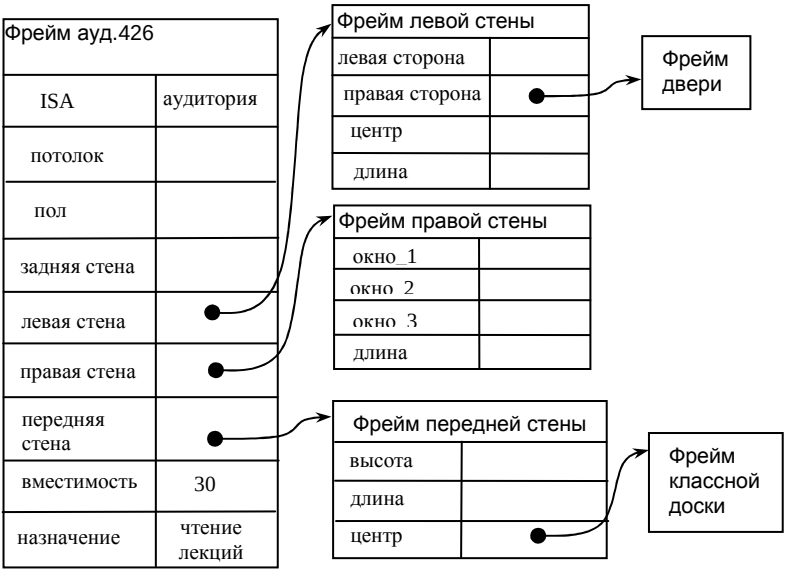


Рис. 4.1

Определение фрейма, данное М. Минским: «Фрейм – это единица представления знаний, запомненная в прошлом, детали которой при необходимости могут быть изменены согласно текущей ситуации» [6].

С помощью фреймов описываются не только статические ситуации (аудитория), но и динамические – празднование дня рождения, подготовка ко дню рождения. Каждый фрейм можно рассматривать как сеть, состоящую из нескольких вершин и отношений (рис. 4.2).

Фрейм прокол	
ISA	Действие
Инструмент	Надутый предмет
Объект	Остроконечный предмет
Способ	(Столкновение инструмент объект)
Результат	(Разрушение объект)
Результат	Взрыв

Рис. 4.2

Значениями слота являются процедуры, параметрами которых являются значения слотов этого же фрейма. Иерархическая структура сети фреймов построена на отношениях «род-вид» и «часть-целое». Отношения «род-вид» характерны тем, что на верхних уровнях расположены абстрактные объекты (концепты), а на нижних уровнях – конкретные объекты. Причем объекты нижних уровней наследуют значения атрибутов объектов верхних уровней. Эти отношения еще называют отношениями типа ISA. Такое название объясняется формой записи «ПРОКОЛ» *is a* «действие».

Отношение «часть-целое» касается структурированных объектов и показывает, что объект нижнего уровня является частью объекта верхнего уровня (стена является структурным элементом аудитории).

Рассмотрим более подробно структуру данных фрейма (табл. 4.1).

Таблица 4.1

Имя фрейма				
Имя слота	Указатель наследования	Указатель атрибутов слота или типа данных	Значение слота	Демон
СЛОТ_1				
СЛОТ_2				
…				
СЛОТ_n				

1. ***Имя фрейма*** – идентификатор, присваиваемый фрейму. Имя должно быть уникальным.

2. ***Имя слота*** – идентификатор, присваиваемый слоту; слот должен иметь уникальное имя во фрейме, к которому он принадлежит. Обычно имя слота не несет никакой смысловой нагрузки, но в некоторых случаях оно может иметь специфический смысл. К таким именам, помимо ISA относятся:

DDESENDANTS –указатель прямого дочернего файла;

FINEDBY – имя пользователя, определяющего фрейм;

DEFINEDON – дата определения фрейма;

MODIFIDON – дата модификации фрейма;

COMMENT – комментарий.

Эти слоты называются системными и используются при редактировании базы знаний и управлении выводом.

3. ***Указатели наследования.*** Эти указатели касаются только фреймовых систем иерархического типа, основанных на «родо-видовых» отношениях. Указатели наследования показывают, какую информацию об атрибутах слотов во фрейме верхнего уровня наследуют слоты с такими же именами во фрейме нижнего уровня. Типичные указатели наследования:

Uniqe *U*: уникальный;

Same *S*: такой же;

Range *R*: установка границ;

Overzide *O*: игнорировать.

U показывает, что каждый фрейм может иметь слоты с различными значениями.

S показывает, что все слоты должны иметь одинаковые значения.

R означает, что значения слотов фрейма нижнего уровня должны находиться в пределах, указанных значениями слотов фреймов нижнего уровня.

O означает, что при отсутствии указания значение слота фрейма верхнего уровня становится значением слота фрейма нижнего уровня, но в случае определенно нового значения значение слотов фреймов нижних уровней указывается в качестве окончательных значений слотов. Одновременно *O* выполняет функции указателей *U* и *S*.

Пример использования указателей исследования приведен на рис. 4.3.

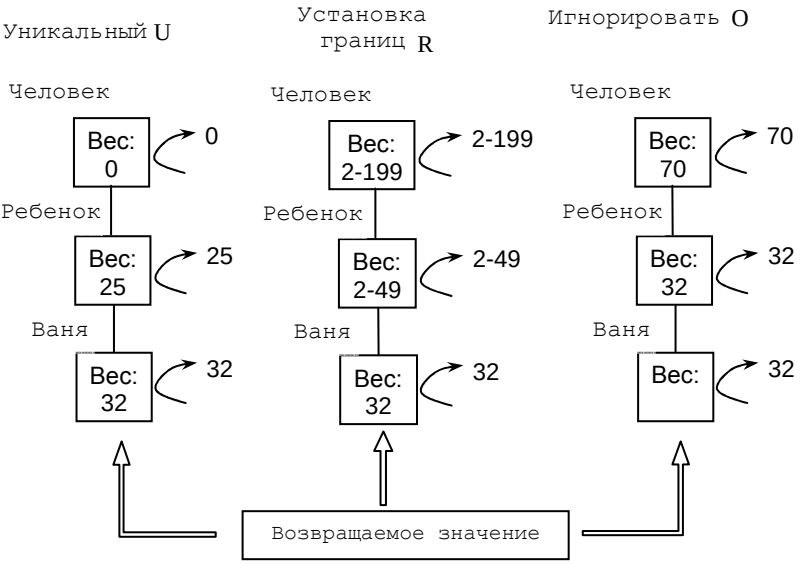


Рис. 4.3

4. ***Указатели типа данных.*** Здесь показывается, что слот имеет численное значение либо служит указателем другого фрейма. К типам данных относятся, например,

FRAME (указатель)	TEXT (текст)
INTEGER (целое)	LIST (список)
REAL (действительное)	TABLE (таблица)
BOOL (булево)	EXPRESSION (выражение)

LISP (присоединятся) процедура и другие.

5. ***Значение слота***. Без комментариев. Единственное, что надо отметить: значение слота должно совпадать с указанным типом данных этого слота; кроме того, должно выполняться условие наследования значение слота: имя, численное значение, процедура, указатель и т.д.

6. ***Демон***. Здесь дается определение демонов типа IF-NEEDE, IF-ADDED, IF-REMOVED и т.д. Демоном называется процедура, автоматически запускаемая при выполнении некоторого условия. Демоны запускаются при обращении к соответствующему слоту. Например, демон FRL IF-NEEDED запускается, если в момент обращения к слоту его значение не было установлено, IF-ADDED запускается при подстановке в слот значения, IF-REMOVED запускается при стирании значения слота. Демон является разновидностью присоединенной процедуры.

7. ***Присоединенная процедура* (*ПП*)**. В качестве значения слота можно использовать программу процедурного типа, называемую служебной. ПП запускается по сообщению, переданному из другого фрейма. Важно отметить, что в языках представления знаний фреймами отсутствует специальный механизм управления выводом, поэтому пользователь должен реализовать данный механизм с помощью присоединенной процедуры.

4.2 Выводы во фреймовых системах

Разделение средств представления знаний и средств вывода во фреймовых системах достаточно затруднительно и носит условный характер. Во фреймовых системах можно выделить три способа управления выводом: два – с помощью присоединительных процедур – *демона* и *служебной процедуры*; один – с помощью *механизма наследования*.

С помощью механизма управления наследованием, базирующегося на отношениях «род-вид» или ISA-отношениях, осуществляется автоматический поиск и определение значений слотов фрейма верхнего уровня и присоединенных процедур служебного типа. С помощью механизма наследования экономится память и сокращается объем работ при программировании. Посредством объединения демона и служебной процедуры можно рационально реализовать любой механизм управления выводом. Однако для согласованного и правильного функционирования системы в целом необходимо ее тщательное проектирование. Здесь необходимо решить проблему зацикливания, которая связана с возможностью авторекурсивного определения фреймов, т.е. с возможностью ссылок фреймов на себя непосредственно или через другие фреймы. Например, демон IF-ADDED в слоте «А» может добавить значение в слот «В», который в свою очередь, добавит значение в «А». Процесс зацикливается.

Для борьбы с зацикливанием в FRL используется комментарий, описывающий источник информации. Доступ к комментарию возможен из любых процедурных средств. Еще одним решением проблемы зацикливания является ограничение на глубину входа в цикл и на общую глубину распространения.

Рассмотрим несколько простых примеров иерархических систем планирования. Эти системы базируются на отношениях ISA между фреймами. Первый пример (см. рис. 4.4) – описание семинара (совещания) в большой компании или фирме [4].

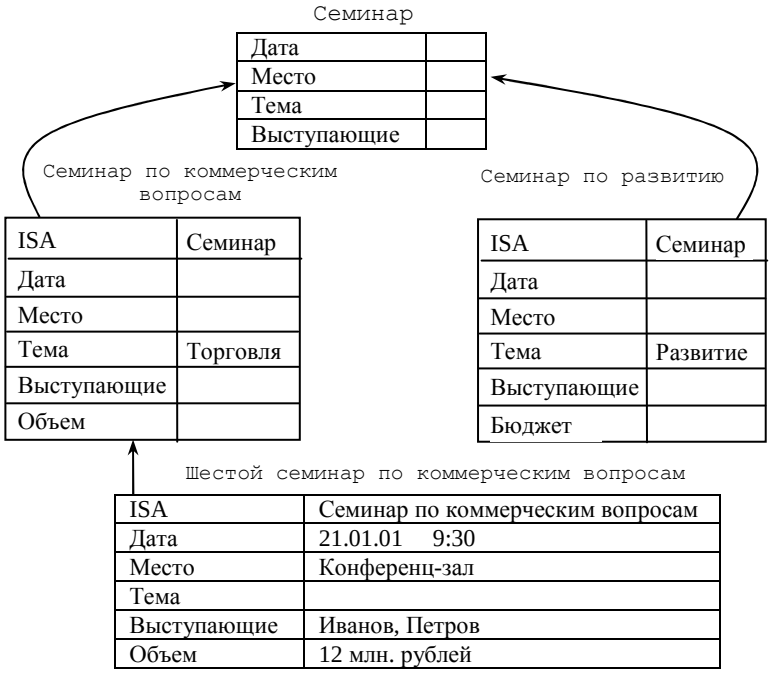


Рис. 4.4

На самом верхнем уровне иерархии определен фрейм «Семинар», содержащий 4 слота. В нашем случае семинары разделяются на семинары по коммерческим вопросам и семинары по развитию. Эти фреймы определены в качестве соответствующих дочерних фреймов. Далее, общей темой семинаров по коммерческим вопросам является торговля, а общей темой семинаров по развитию является освоение новых видов продукции, которые подставляются в качестве значений в соответствующие слоты. Кроме того, во фрейме «Семинар по коммерческим вопросам» задается бюджет на освоение. В нашем примере в качестве фрейма, дочернего по отношению к фрейму «Семинар по коммерческим вопросам», определен фрейм «Шестой семинар по коммерческим вопросам», в четыре слота которого введены конкретные значения. Исключение составляет слот «Тема». Что касается слота «тема», то здесь в момент обращения к слоту «Тема» фрейма «Шестой семинар по коммерческим вопросам» запускается механизм управления наследованием, с помощью которого осуществляется поиск и под-

становка значения этого слота в соответствующем фрейме. Причем поиск ведется и во фреймах верхнего уровня.

Рассмотрим подробнее структуру фрейма «Шестой семинар по коммерческим вопросам» (рис. 4.5).

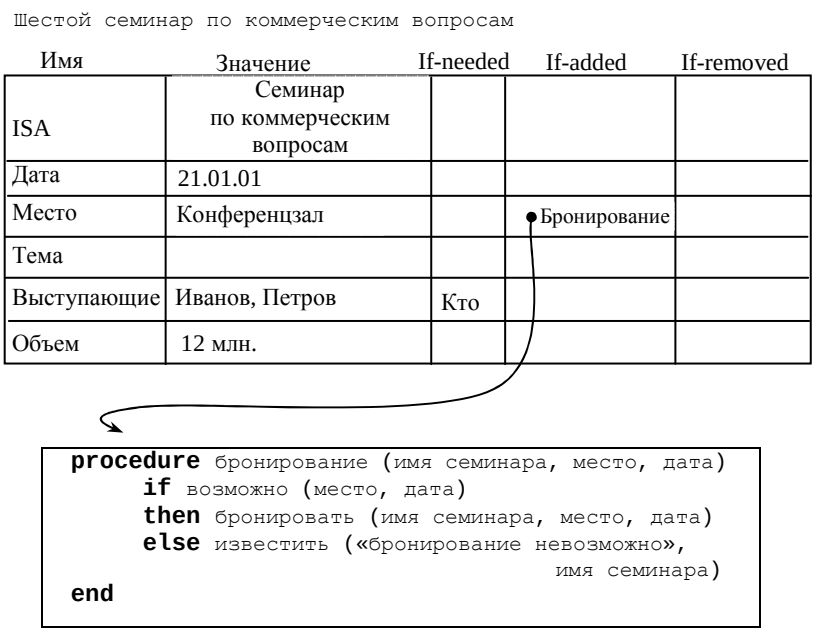


Рис. 4.5

В нашем случае в каждом слоте можно указать три типа демонов. В общих чертах демон функционирует следующим образом. В данном примере в слоте «Место» определен демон IF-ADDED с именем «Бронирование», который автоматически запускается при подстановке в этот слот значения «Конференц-зал». Если этот зал можно занять, то он бронируется, в противном случае, когда зал уже занят, выдается сообщение «Бронирование невозможно». Демон IF-NEEDED с именем «Кто», присоединенный к слоту «Выступающие», в случае, если при обращении к данному слоту его значение не было определено, генерирует запрос: «Кто выступает на шестом семинаре по коммерческим вопросам?» Ответ на этот вопрос передается при подстановке входных данных пользователя в качестве значения слота.

Рассмотрев работу демонов, перейдем к рассмотрению работы служебной процедуры на примере, приводимом на рис. 4.6.

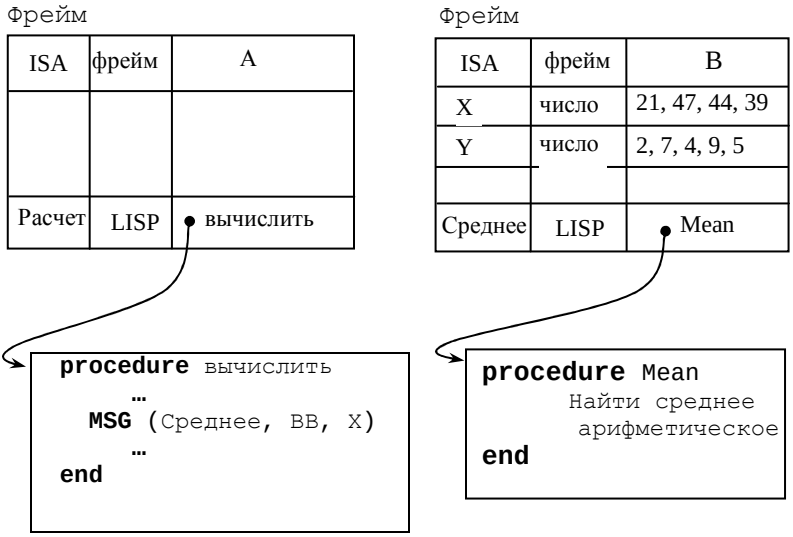


Рис. 4.6

Команда MSG, имеющаяся в присоединенной процедуре с именем «Вычислить», относящаяся к слоту «Расчет» фрейма «АА», служит для передачи сообщений. Структура предложения этой команды выглядит следующим образом:

MSG(имя фрейма, имя слота, параметр,...)

Механизм передачи/приема сообщений с помощью команды MSG действует следующим образом. Если по сообщению от другого фрейма инициируется присоединенная процедура «Вычислить» фрейма «АА», то с помощью команды MSG фрейму «ВВ» передается сообщение, по которому инициируется присоединенная процедура MEAN слота «Среднее» фрейма «ВВ». С помощью этой процедуры вычисляется среднее арифметическое четырех значений 21, 47, 44, 39 слота «X». Результат вычисления «37,75» передается во фрейм «АА».

Использование фреймовых систем целесообразно в тех проблемных областях, в которых важную роль для решения задач играют ожидаемые результаты относительно вида и содержания данных, например, при интерпретации визуальных сцен или распознавании речи. Так, в системах, работающих по запросам на ЕЯ, должны обеспечиваться понимание смысла входной фразы и формирование правильно-

го ответа. Это достигается путем механизмов сопоставления фреймов, построенных на запросах, с фреймами базы знаний и получения в результате этого фреймов, в которых все слоты заполнены.

Во фреймовых системах отдается предпочтение режиму вывода перед интерактивным режимом, т.е. делается все возможное, чтобы извлечь информацию из базы знаний, и только в случае неудачи система связывается с пользователем для получения этих данных.

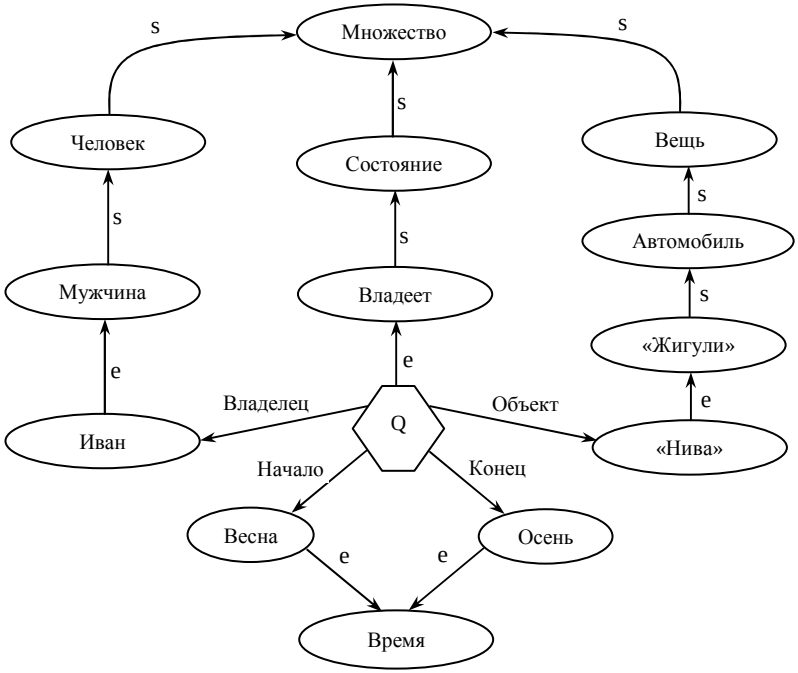
Контрольные вопросы

1. Определите характер описания информации во фреймовых представлениях.
2. Опишите структуру данных фрейма в языке FRL.
3. Назовите условия запуска демонов и присоединенных процедур.
4. Разработайте фреймы для описания посещения дня рождения.

5 СЕМАНТИЧЕСКИЕ СЕТИ (СС)

5.1 Модели семантических сетей

Стандартного определения семантической сети не существует, но обычно под ней подразумевают систему знаний, имеющую определенный смысл в виде целостного образа сети, узлы которой соответствуют понятиям и объектам, а дуги – отношениям и связям [6]. Например, если «Иван» и «Мальчик» являются узлами сети, то, установив между этими узлами связь «Есть», получим смысловое предположение «Иван – это мальчик». Рассмотрим еще пример «Иван с весны до осени владел автомобилем марки "Нива"» (рис. 5.1).



S – подмножество;
e – элемент.

Рис. 5.1

Идея использования СС для запоминания сложных понятийных структур возникла первоначально в психологии. Существует большое количество различных методов построения СС от простейших, используемых в работах по психологии, до сложных методов, мало отли-

чающихся от сетей фреймов. Однако в большинстве моделей стараются сохранить основной принцип неразделимости синтаксических, или структурных, и семантико-прагматических знаний о внешнем мире.

В качестве модели СС рассмотрим модель, получившую название TLC-модели (Teachable Language Comprehender: доступный механизм понимания языка). Здесь основой для определения значения того или иного понятия является множество взаимосвязей с другими понятиями (т.е. определение понятия дается через отношения с другими понятиями). Среди этих связей основными являются следующие:

- класс, к которому принадлежит данное понятие;
- свойства, выделяющие понятие из всех прочих понятий этого класса;
- примеры данного понятия.

В качестве примера покажем простейшую сеть для представления концептуального объекта «чайник» (рис. 5.2).

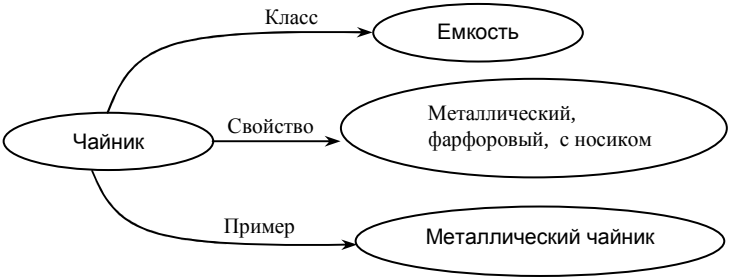


Рис. 5.2

Здесь с отношением «Класс» ассоциируется значение емкость, со свойством – «Металлический», «Фарфоровый», «Наличие носика», с примером – «Металлический чайник».

Рассмотрим еще один пример сети для представления концептуального объекта «Автомобиль» (рис. 5.3).

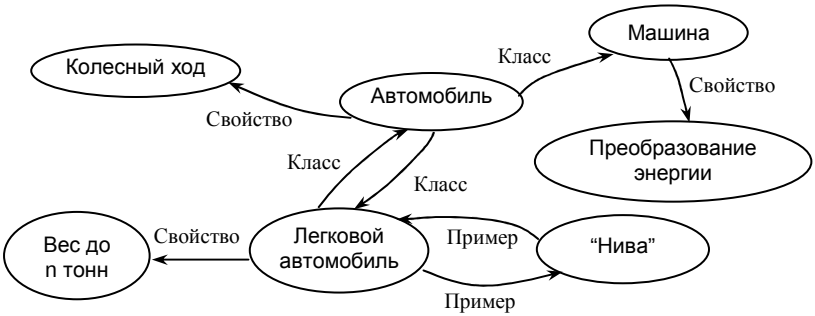


Рис. 5.3

Отношения, определяющие сущности, имеющие место в реальном мире, могут быть расширены. Здесь, помимо классов, свойств и примеров, могут быть использованы следующие отношения: «Время», «Место», «Средство», «Объект» и т.п.

Модель TLC получила развитие в работах многих авторов, которые внесли свои существенные изменения. Так например, вместо связи «Класс» использовалась связь «Есть нек.» (есть некоторый), которая связывает предмет с более общим понятием. Вместо связи «Свойство» использовались два глагола «Имеет» и «Есть». Глагол «Имеет» употребляется в тех случаях, когда свойством оказывается наличие некоторого предмета или владение им, например, птица имеет крылья. Глагол «Есть» используется тогда, когда свойство имеет характер качества, например, «Канарейка есть желтая». Заметим, что «Примерам» почти всегда соответствуют имена классов, т.е. «Класс» и «Пример» связывают одни и те же понятия, но направлены противоположно друг другу. Поэтому связь «Пример» может быть исключена. Пример сети для представления концептуального объекта «Животное» приведен на рис. 5.4.

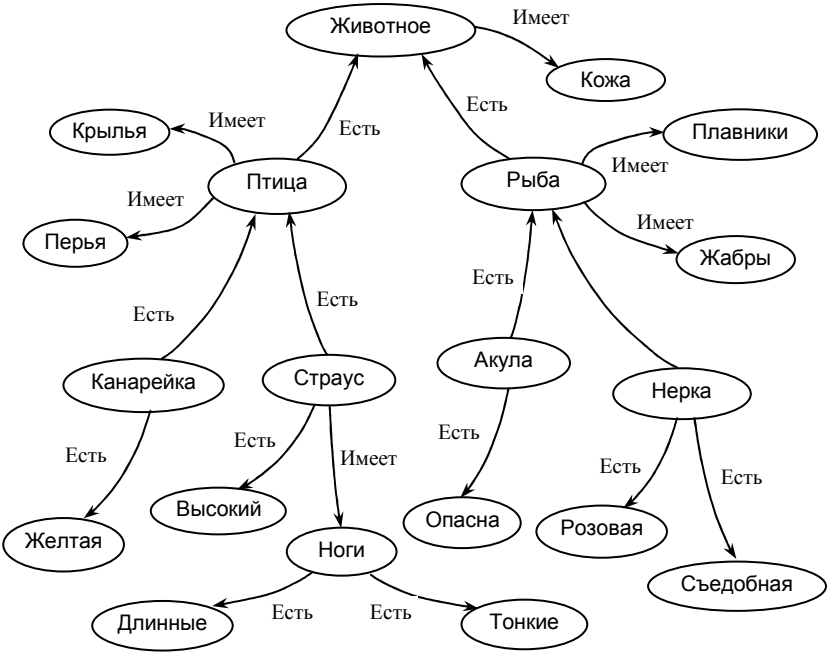


Рис. 5.4

Можно вводить еще отношения, например «Может» указывает характерное, но необязательное свойство, например, птица может летать, животное может двигаться и т.д.

При рассмотрении способов представления разных видов информации в виде понятийных структур может возникнуть необходимость использования многих примеров одного и того же понятия, причем в каждом конкретном случае понятие может выступать в видоизмененной форме. Поэтому целесообразно ввести первичное и вторичное определения понятия. Под первичным определением понимается первое, основное, определение. Другие же конкретные случаи использования этого понятия связаны с первичным понятием связью «Есть нек.» и определяются своими свойствами (см. пример на рис. 5.5).

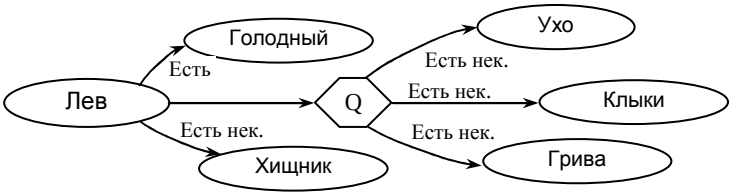


Рис. 5.5

Здесь узел Q через связи «Есть нек.» может в различном контексте представляться в виде уха, гривы и клыков. Это понятие созвучно понятию слотов и наследования во фреймовых моделях.

Рассмотрим теперь средства, дающие возможность представлять в модели события и действия. Основная цель здесь состоит в выделении набора простых отношений, характеризующих основные компоненты события. В лингвистике различаются два основных уровня языка. На уровне поверхностных структур представляются синтаксические структуры предложения. На другом уровне, уровне глубинных структур, представляется смысловое содержание предложений.

Для обнаружения структуры события в предложении в первую очередь выделяют само действие, описываемое обычно глаголом. Далее необходимо найти лиц, которые действуют, и выделить предметы, над которыми это действие производится. Действующее лицо, осуществляющее действие, называется «Агент». Вещи, над которыми действие осуществляется, называются «Объект». Лицо, пользующееся результатом действия или испытывающее его, называется «Адресат». Пример: «Мать прибирает комнату сына».

Действие: уборка.
Агент: мать.
Объект: комната.
Адресат: сын.

На рис. 5.6 приведено графическое представление результатов анализа этого предложения.

Рис. 5.6

Эти структуры отображают смысл предложения и являются глубинными. Поэтому предложение «Комната сына прибирается матерью» имеет то же графическое представление. В этом проявляется важное преимущество структурных методов СС, когда один и тот же смысл или семантика представляются одной и той же структурой.

В рассмотренном примере использовались только три отношения: агент, объект и адресат, называемые падежами или падежными отношениями (падежами Филмора), кроме них употребляются такие, как:

- «Условие» – логическая зависимость, существующая между двумя событиями: «Акула опасна, только если она голодная»;
- «Инструмент» – предмет или устройство, вызывающее действие или являющееся орудием его осуществления: «Ураган разрушил дом»;
- «Место» – указание на то, где происходит событие: «Из института он поехал на такси»;
- «Модальность» – указание в случае необходимости на особый вид поверхностной структуры: «Почему Вы туда поехали?» (модальность вопросительная);
- «Цель» – указание на цель действия: «Иван повернул стопкран, чтобы остановить поезд»;
- «Качество» – ограничение объема понятия: «Шторм был сильным»;
- «Время» – указание на то, когда происходит событие: «Вчера был шторм».

Пример: «Вчера на берегу я фотографировал своим новым аппаратом дом на набережной».

Анализ:

Действие: фотографировать.
Агент: Я.
Объект: дом на набережной.
Место: берег.

Инструмент: мой новый аппарат.

Время: вечер.

Графическое представление результатов анализа приведено на рис. 5.7.

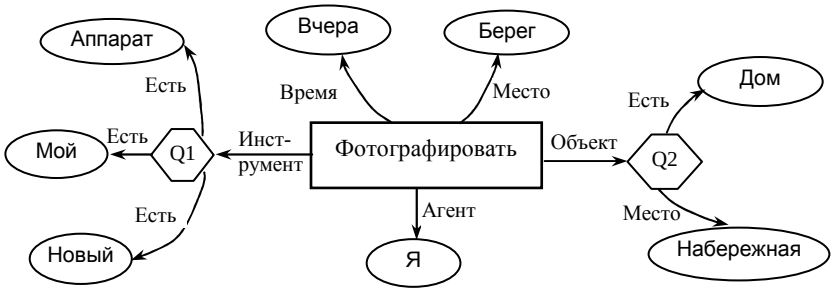
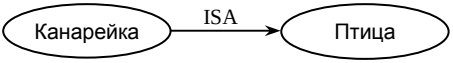


Рис. 5.7

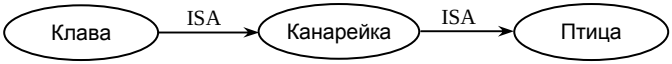
5.2 Выводы в семантических сетях

В иерархической структуре понятий существуют отношения, по крайней мере, двух типов: ISA – «род-вид» и has-part – «часть-целое». Иерархия показывает отношения включения понятия. Например, в предложении «Человек ISA млекопитающее» используется отношение включения «род-вид». При этом экземпляр нижнего уровня содержит в основном все атрибуты, которые имеет экземпляр понятия верхнего уровня. Это свойство называется наследованием атрибутов между уровнями иерархии ISA.

Отношение «has-part» в предложении «рыба has-part плавники», показывает, что экземпляр «Плавники» является частью любого экземпляра «Рыба». Этот способ показывает отношения между экземплярами класса. Рассмотрим предложение «Все канарейки – птицы». Графическая форма приведена ниже:



Если канарейке присвоить имя «Клава», то сеть расширится следующим образом:



В данном случае, вместе с тем, что с помощью такой сети представлены два факта «Клава – канарейка» и «Канарейка – птица», из нее, используя отношение ISA, можно вывести факт «Клава – птица».

Расширим нашу сеть фактом «Птицы имеют крылья» (рис. 5.8).

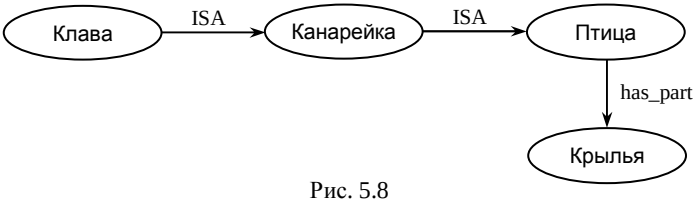


Рис. 5.8

Используя отношения ISA и has part, можно вывести факт «Клава имеет крылья». Этот способ вывода называются *наследованием* свойств. Еще одно расширение сети фактами «Канарейка – вид, подвергающийся опасности, изучается натуралистами» (рис. 5.9):

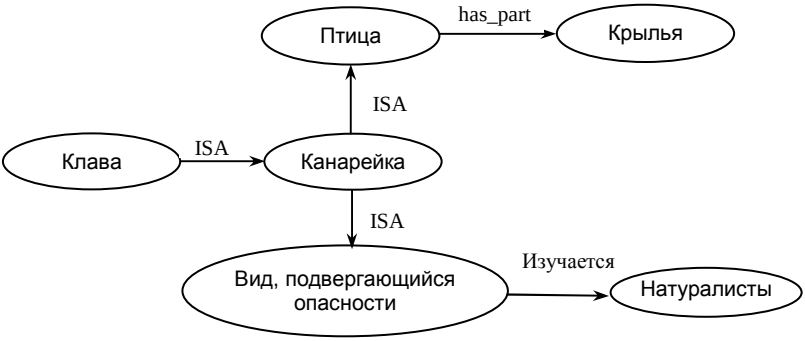


Рис. 5.9

В сети, указанной на рис. 5.9, можно вывести заключение, что канарейка по имени Клава изучается натуралистами. Однако это заключение необязательно является правильным. Следовательно, в СС необходимо предусмотреть такие способы представления данных и способы вывода, которые обеспечивали бы одновременно управление наследованием.

Особенность СС, которая в то же время является ее недостатком, заключается в целостности системы, выполненной на ее основе, не позволяющей разделить базу знаний и механизм выводов. Обычно интерпретация СС определяется с помощью использующих ее процедур. Типичным способом, лежащим в основе этих процедур, является способ *сопоставления частей сетевой структуры*. Этот способ основан на построении подсети, соответствующей вопросу, и сопоставлении ее с базой знаний. При этом для исчерпывающего сопоставления с базой знаний вершинам переменных подсети присваиваются гипотетические значения. Например, пусть мы в базе знаний имеем сеть, содержащую факт «Канарейка Клава владеет гнездом» (рис. 5.10).

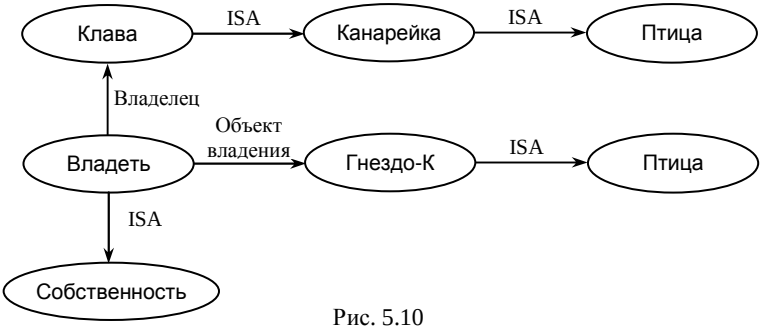


Рис. 5.10

Здесь «Гнездо-К» – это конкретное гнездо, которым владеет Клава, оно является экземпляром понятия «Гнездо». Вопрос «Чем владеет Клава?» представляется в виде следующей подсети (рис. 5.11).

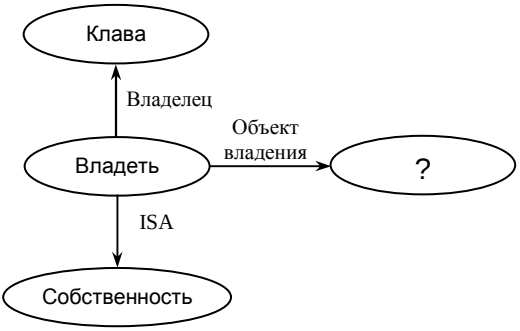


Рис. 5.11

Проводится сопоставление для ответа на вопрос. При этом сначала отыскивается вершина «Владеть», имеющая ветвь «Владелец», направленную в вершину «Клава», затем следует соединение с вершиной, которая показывает ветвь «Собственность» и ответ на вопрос.

Рассмотрим еще один вопрос «Существует ли птица, которая владеет гнездом?» Подсеть вопроса имеет вид, изображенный на рис. 5.12.

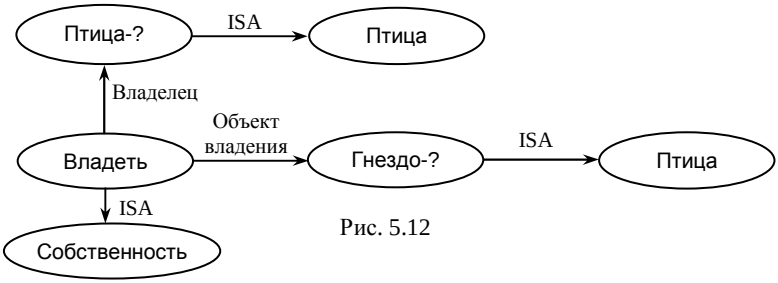


Рис. 5.12

Однако эта подсеть не годится для непосредственного сопоставления с базой знаний. Для сопоставления из узла «Клава» к узлу «Птица» проводится дуга ISA, означающая, что Клава является птицей. Теперь становится возможным сопоставление «Владеть» – Владеть», «Гнездо-?» – «Гнездо-К», «Гнездо» – Гнездо». Ответ: «Да – это Клава».

Рассмотрим способ вывода, именуемый *перекрестным поиском*. Этот способ означает поиск отношений между концептуальными объектами и ответ на вопрос путем обнаружения узла, в котором пересекаются дуги, идущие от двух различных узлов. Например, пусть дан факт «Иван дал книгу Петру». Сеть, представляющая данный факт, приведена на рис. 5.13.

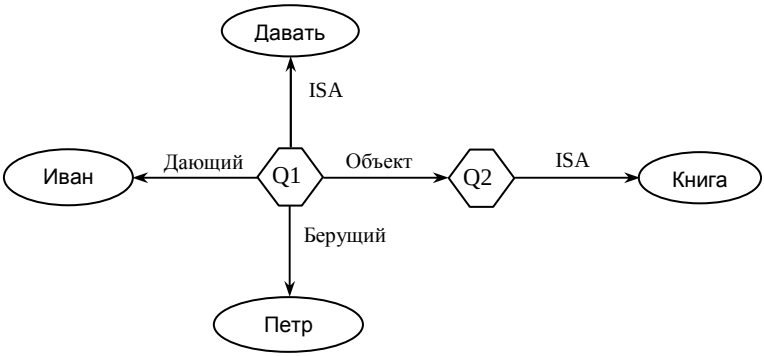


Рис. 5.13

На вопрос «Какие отношения между Иваном и Петром» можно получить ответ «Иван дал книгу Петру».

Рассмотрим семантическую сеть специального вида, носящую название «Функциональная семантическая сеть» (ФСС) [2]. Она представляется в виде графа с вершинами двух типов. Вершинам одного типа соответствуют различные параметры, участвующие в решении задачи. Эти параметры могут либо задаваться как исходные данные, либо вычисляться по ходу выполнения будущей программы решения задач. Вершинам второго типа соответствуют функциональные отношения, связывающие между собой эти параметры. Поясним, как используется ФСС при решении конкретных задач, связанных с планированием вычислений. Пусть проблемная область – это планиметрия треугольника. Фрагмент ФСС, соответствующий этой проблемной области показан на рис. 5.14.

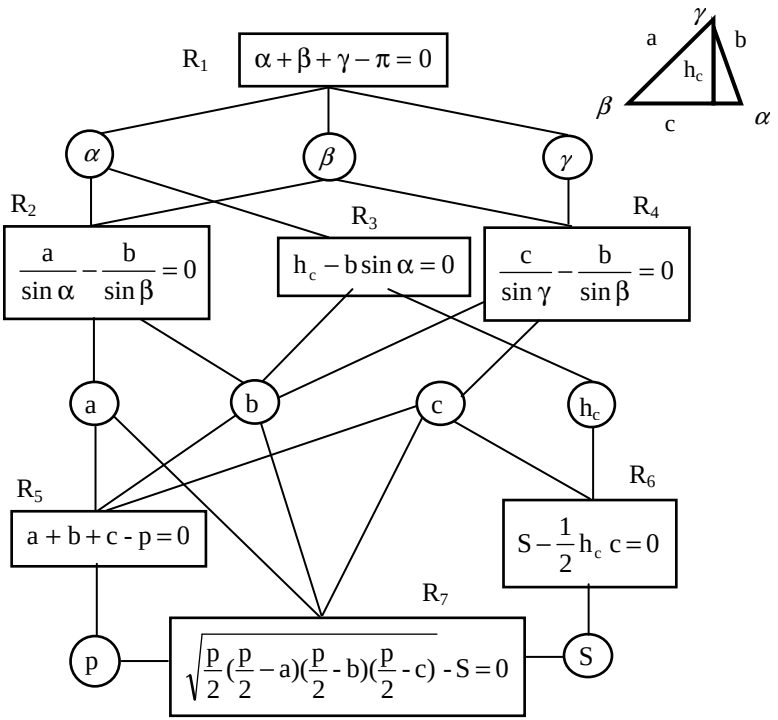
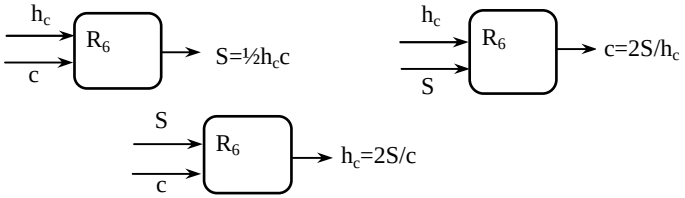


Рис. 5.14

Здесь кружки соответствуют параметрам, а прямоугольники – функциональным отношениям между ними. Будем считать, что все отношения имеют разрешение относительно всех входящих в них параметров. Каждому такому разрешению соответствует специальный программный модуль. Так например, отношение R_6 будет иметь три различных разрешения, показанных ниже:



Пусть пользователь задумал решить задачу: «Найти площадь треугольника *S* по известной стороне *c* и известным прилегающим углам α и β».

Известны два метода вывода или планирования в ФСС. Первый – метод распространяющейся волны. В наиболее простой форме он может быть описан следующим образом. По условию задачи c, α и β известны. Тогда в ФСС как бы возбуждаются вершины, соответствующие этим параметрам. Это возбуждение распространяется от них по ребрам к вершинам-прямоугольникам. Условием возбуждения этих вершин является возбуждение $(m-1)$ ребер, подходящих к ним, где m – число ребер, подходящих к вершинам-прямоугольникам. Возбуждение вершины-прямоугольника соответствует разрешению отношения, написанного внутри прямоугольника относительно параметра, которому соответствует невозбужденное ребро. В нашем примере возбуждение вершин c, α и β приведет на следующем шаге к возбуждению вершины-прямоугольника R_1 и разрешению относительно γ. Далее возбуждение от вершин c, γ и β будет передано к вершине-прямоугольнику R_4, которая возбудится, дав разрешение относительно b. После этого произойдет возбуждение вершины-прямоугольника R_3, которая даст разрешение относительно h_c, и, наконец, возбудится вершина-прямоугольник R_6, дав разрешение относительно S. Возбуждение вершины, соответствующей S, прекратит распространение волны.

Рассмотрим алгоритм паросочетания для планирования вычислений в ФСС. Этот алгоритм хорошо известен в теории графов. С его помощью находится взаимно-однозначное соответствие между вершинами, соотносимыми с параметрами, и вершинами-прямоугольниками. Граф для решения нашей задачи имеет вид, представленный на рис. 5.15.

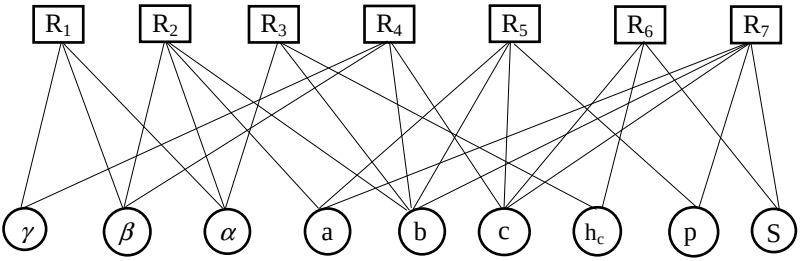


Рис. 5.15

После первого этапа работы алгоритма, когда найден искомый параметр, надо найти минимальную замкнутую систему отношений, позволяющую решить поставленную пользователем задачу. Так как в нашем случае определять a и периметр треугольника p нет необходимости, то минимальной замкнутой системой является система отношений: R_1, R_4, R_3, R_6.

Контрольные вопросы

1. Назовите основные отношения, принятые в TLC-модели.
2. Что такое первичное и вторичное определения понятия при представлении разных видов информации в виде понятийных структур?
3. Определите способы вывода в семантических сетях. Приведите примеры.
4. Что такое падеж Филмора? Какую информацию можно представить с помощью таких падежей?
5. Выясните отличия семантической сети общего вида от функциональной семантической сети.

6 НЕЧЕТКИЕ ЗНАНИЯ

6.1 Нечеткие множества

Знания не всегда могут быть описаны точно – часто встречаются так называемые «нечеткие знания». Люди повседневно решают проблемы и делают заключения в среде «нечетких знаний», а для того чтобы интеллектуальные системы обладали такими возможностями как гибкость, широкий кругозор, адаптируемость, необходимо представление и использование нечетких знаний. Все нечеткости, с которыми имеет дело инженерия знаний, можно классифицировать следующим образом:

1) недетерминированность выводов;
2) неоднозначность (зашумленный сигнал);
3) ненадежность (в силу ограниченности точности прибора);
4) неполнота (пропуск значений в таблице факторы/испытания);
5) собственно нечеткость (лингвистические аспекты языка).

Методологической основой для формализации нечетких знаний является теория нечетких множеств [5].

Когда мы говорим «Старик», то неясно, что мы имеем в виду: старше 50, старше 60 или старше 70? Одним из методов изучения множеств без уточнения их границ является теория нечетких множеств, которая была предложена Л. Заде в 1965 г. и продолжает развиваться. Эти исследования связаны также с нечеткими выводами, которые выполняются с использованием правил, представленных как нечеткие множества. В данном разделе мы ознакомимся с основными понятиями теории нечетких множеств и методами нечетких выводов.

Определение

Нечеткое подмножество F множества элементов U определяется функцией принадлежности $\mu_F(u)$. Эта функция отображает элементы u множества U на множество чисел в интервале [0,1], которые указывают степень принадлежности каждого элемента нечеткому подмножеству F.

Если множество U состоит из конечного числа элементов $\{u_1, u_2, ..., u_n\}$, то нечеткое подмножество F можно представить следующим образом:

$$F = \mu_F(u_1)/u_1 + \mu_F(u_2)/u_2 + ... + \mu_F(u_n)/u_n = \sum_{i=1}^{n} \mu_F(u_i)/u_i.$$

Следует иметь в виду, что знак плюс в этой формуле означает не суммирование, а объединение или конъюнкцию, а символ « / » показывает, что значение $\mu_A(u_i)$ относится к элементу, следующему за ним.

Пример

U = 1 + 2 + 3 + ... + 10. Тогда нечеткое множество A, которое описывается понятием «несколько», можно записать в следующем виде: «несколько» = 0,4/2 + 0,7/3 + 0,8/9, символ «=» означает здесь равенство по определению.

В случае непрерывного множества U вводится следующее обозначение подмножества F: $\int_U \mu(u)/u$.

Следует иметь в виду, что знак $\int$ в приведенной выше формуле означает не интегрирование, а объединение. Заметим также, что если $\mu_A(u)$ принимает значение только 0 или 1, то множество A является обычным множеством. Запись $\mu_A(u)=1$ означает, что элемент $u \in U$ принадлежит множеству A, т.е. $u \in A$. Запись $\mu_A(u)=0$ означает, что $u \in U$ не принадлежит множеству A, т.е. $u \notin U$.

Пусть U – множество людей в возрасте 0–100 лет, функции принадлежности нечетких множеств, означающих возраст: «Молодой», «Средний», «Старый» можно определить при помощи графика (рис. 6.1).

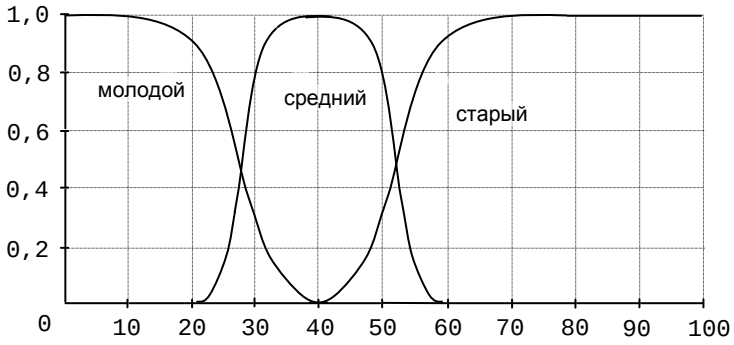


Рис. 6.1

При дискретизации через 10 лет получим приблизительно следующее:

молодой = 1/0 + 1/10 + 0,9/20 + 0,3/30,

средний = 0,8/30 + 1/40 + 0,8/50,

старый = 0,3/50 + 0,9/60 + 1/70 + 1/80 + 1/90+ 1/100.

Здесь элементы множества с функцией принадлежности, равной 0, не записываются.

6.2 Операции на нечетких множествах

Над нечеткими множествами выполняются те же операции, что и над обычными множествами.

Понятие нечеткого подмножества Л. Заде определил следующим образом: нечеткое подмножество данного конечного множества U – это такое подмножество, значения степеней принадлежности элементов которого лежат в единичном интервале [0,1]. Пусть $A = \int_U \mu A(u)/u$ и $B = \int_U \mu B(u)/u$ – два нечетких множества, тогда нечеткое множество A является *подмножеством* нечеткого множество $B (A \subseteq B)$, если для всех u справедливо неравенство $\mu_A(u) \le \mu_B(u)$.

Равенство двух нечетких множеств определяется следующим образом: два нечетких множества A и B равны $(A = B)$, если для всех u справедливо $\mu_A(u) = \mu_B(u)$.

ОБЪЕДИНЕНИЕ нечетких множеств *A* и *B*:

$$A \cup B = \int_U \left(\mu A(u) \vee \mu B(u)\right)/u,$$

ПЕРЕСЕЧЕНИЕ нечетких множеств *A* и *B*:

$$A \cup B = \int_U \left(\mu A(u) \wedge \mu B(u)\right)/u,$$

где $\left(\mu A(u) \wedge \mu B(u)\right) = \min \mu A(u), \mu B(u)$.

ДОПОЛНЕНИЕ или *ОТРИЦАНИЕ* определяется следующей формулой: $\sim A = \int_U \left(1 - \mu_A(u)\right)/u$.

Квантификатор **не** может интерпретироваться с помощью операции отрицания:

$$\textbf{не}\ \ x = \left(1 - \mu_x(u)\right)/u.$$

РАЗНОСТЬ двух нечетких множеств определяется формулой

$$A - B = A \cap \sim B.$$

ДИЗЪЮНКТИВНАЯ СУММА определяется соотношением

$$A + B = (A \cup B) - (A \cap B).$$

Например,

~(*молодой*) = 0,1/20+0,7/30+1/40+1/50+1/60+1/70+1/80+1/90+1/100.

(*молодой*) ∪ (*средний*) = 1/0+1/10+0,9/20+0,8/30+1/40+0,8/50.

(*молодой*) ∩ (*средний*) = 0,3/30.

ПРОИЗВЕДЕНИЕ нечетких множеств A и B определяется следующим соотношением: $A \cdot B = \int_U \left(\mu_A(u) \cdot \mu_B(u)\right)/u.$

Операция *возведения в степень* $A^b = \int_U \left(\mu_A(u)\right)^b / u,$ где $b > 0.$

Операция *концентрирования* нечеткого множества CON(A) = A^2. Эта операция уменьшает степень принадлежности элементов тем больше, чем меньше степень их принадлежности первоначальному множеству A. Квантификатор **очень** может интерпретироваться с помощью операции концентрации, то есть возведения в квадрат:

$$\textbf{очень}\ \ x = \int \mu_x^2(u)/u.$$

Операция *растяжения* нечеткого множества является противоположной концентрации и определяется соотношением $DIL(A) = A^{0,5}$.

Операция контрастной интенсивности определяется соотношением $INT(A) = \begin{cases} 2 \cdot A^2, \ 0 \le \mu_A(u) \le 0.5; \\ \sim \left(2 \cdot (\sim A)^2\right), \ 0.5 \le \mu_A(u) \le 1. \end{cases}$

Эта операция увеличивает значения $\mu_A(u)$, которые больше 0,5, и уменьшает те значения $\mu_A(u)$, которые меньше 0,5, уменьшая тем самым нечеткость A.

6.3 Нечеткие отношения

Пусть U и V – универсальные множества, на которых определены X и Y соответственно, тогда нечеткое отношение R: $X \to Y$ определяется как подмножество декартова произведения двух нечетких множеств $X \times Y \subseteq U \times V$, которое задается с помощью функции принадлежности двух переменных по формуле $R = \int_{X \times Y} \mu_R(u,v)/(u,v).$

В общем случае n-арное отношение, или n-отношение, определяется следующим образом. Пусть R – результирующее множество декартова произведения n множеств и μ – его функция принадлежности. Нечеткое n-отношение определяется как нечеткое подмножество R, принимающее какое-либо значение на интервале функции принадлежности в соответствии со следующей формулой:

$$R = \int_{X_1 \times ... \times X_n} \mu_R(u_1, ..., u_n)/(u_1, ..., u_n).$$

$x_i \in X_i,\ i = 1, ..., n.$

Пример

Допустим, что X = {Иван, Марья}; Y = {Петр, Дарья}, тогда

Дружба = 0.6/(Иван, Петр) + 0.9/(Иван, Дарья) +
+ 0.8/(Марья, Петр) + 0.1/(Марья, Дарья).

Отношения удобно записывать с помощью матрицы отношений

	Петр	Дарья
Иван	0.6	0.9
Марья	0.8	0.1

Допустим, что существует нечеткое знание-правило типа

если F, то G

(если старый, то умный), использующее нечеткие множества $F \subset U$ и $G \subset V$. Тогда один из способов построения нечеткого отношения из соответствующей области полного множества U в область полного множества V состоит в следующем:

$$R = F \times G = \int_{U \times V} \left(\mu_F(u) \wedge \mu_G(v)\right)/(u, v)$$

или

$$R = F \times G = \sum_i \sum_j \left(\mu_F(u_i) \wedge \mu_G(v_j)\right)/(u_i, v_j).$$

Необходимо отметить, что есть и другие способы построения нечеткого отношения.

Пусть U и V – это области натуральных чисел от 1 до 4, тогда определим следующим образом нечеткие множества:

F = маленькие = 1/1 + 0.6/2 + 0.3/3 + 0/4,

G = большие = 0/1 + 0.1/2 + 0.6/3 + 1/4.

Если есть нечеткое знание-правило

если u – маленькое, то v – большое,

то можно следующим образом построить нечеткое отношение, определяющее данное знание-правило:

$$R = F \times G = \begin{vmatrix} 0 & 0.1 & 0.6 & 1 \\ 0 & 0.1 & 0.6 & 0.6 \\ 0 & 0.1 & 0.3 & 0.3 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

Пусть R – нечеткое отношение из области U в область V, S – нечеткое отношение из области V в область W, тогда нечеткое отношение из области U в область W определяется как свертка max-min:

$$R \bullet S = \sum_{i=1}^{n} \sum_{k=1}^{l} \bigcup_{v_j \in V} \left(\mu_R(u_i, v_j) \wedge \mu_S(v_j, w_k)\right)/(u_i, w_k).$$

Здесь знак « $\bullet$ » обозначает свертку max-min,

$\bigcup_{v_j}$ – взятие максимума для всех v_j;

$\wedge$ – взятие минимума.

Пример

$U = V = W = \{1, 2, 3, 4\}$.

F = маленькие = 1/1 + 0.6/2 + 0.3/3 + 0/4,

G = большие = 0/1 + 0.1/2 + 0.6/3 + 1/4.

$\sim F$ = не_маленькие = 0/1 + 0.4/2 + 0.7/3 + 1/4,

G^2 = очень_большие = 0/1 + 0 .01/2 + 0.36/3 + 1/4 или, округлив, 0/1 + 0 /2 + 0.4/3 + 1/4. Тогда если есть знание-правило

если v – не маленькое, то w – очень большое,

то в соответствии с формулой $S = \sim F \times G^2$ можно построить нечеткое отношение S из V в W

$$S = \sim F \times G^2 = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0.4 \\ 0 & 0 & 0.4 & 0.7 \\ 0 & 0 & 0.4 & 1 \end{vmatrix}$$

И далее можно построить нечеткое отношение из U в W.

$$R \bullet S = \begin{vmatrix} 0 & 0.1 & 0.6 & 1 \\ 0 & 0.1 & 0.6 & 0.6 \\ 0 & 0.1 & 0.3 & 0.3 \\ 0 & 0 & 0 & 0 \end{vmatrix} \bullet \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0.4 \\ 0 & 0 & 0.4 & 0.7 \\ 0 & 0 & 0.4 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0.4 & 1 \\ 0 & 0 & 0.4 & 0.6 \\ 0 & 0 & 0.3 & 0.3 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

6.4 Вывод на нечетких знаниях

Традиционный дедуктивный вывод, называемый также *модус поненс*, записывается следующим образом:

$P \Rightarrow Q$
P

Q

Что означает вывод Q из факта P по правилу $P \Rightarrow Q$.

Используя те же обозначения, можно определить нечеткий дедуктивный вывод следующим образом:

$P \Rightarrow Q$
P'

Q'

Однако эта формулировка имеет существенное отличие от традиционного *модус поненс*. Здесь не требуется совпадения высказывания P' в факте и высказывания P в правиле. В общем случае могут не совпадать и заключения Q и Q'.

Л. Заде предложил нечеткий условный вывод в следующей форме:

если x есть A, то y есть B, иначе y есть C
x есть A'

---

y есть D

Здесь x, y – имена объектов; A, A', B, C, D – нечеткие понятия, представленные нечеткими множествами, определенными на U, U, V, V, V соответственно.

Предложено несколько правил, переводящих нечеткое условное высказывание «если x есть A, то y есть B, иначе y есть C» в нечеткое отношение $U \bullet V$.

Пусть A, B, C – нечеткие множества в U, V, V, заданные в виде

$$A = \int_U \mu A(u)/u;\ B = \int_V \mu B(v)/v;\ C = \int_V \mu C(v)/v.$$

Тогда имеем:

А. Максиминное правило Rm':

$$Rm' = (A \times B) \cup (\sim A \times C) =$$

$$= \int_{U \times V} \left(\mu A(u) \wedge \mu B(v)\right) \vee \left(1 - \mu A(u)\right) \wedge \mu C(v)/(u, v).$$

Б. Арифметическое правило Ra':

$$Ra' = (\sim A \times V + U \times B) \cap (A \times V + U \times C) =$$

$$= \int_{U \times V} 1 \wedge \left(1 - \mu A(u) + \mu B(v)\right) \wedge \left((\mu A(u) + \mu C(v)\right)/(u, v).$$

В. Размытое бинарное правило

$$Rb' = (\sim A \times V \cup U \times B) \cap (A \times V \cup U \times C) =$$

$$= \int_{U \times V} 1 \wedge \left(1 - \mu A(u) \vee \mu B(v)\right) \wedge \left((\mu A(u) \vee \mu C(v)\right)/(u, v).$$

Г. Правила Танака-Мидзумото [4, 10]

$$Rgg' = (\sim A \times V \Rightarrow U \times B) \cap (A \times V \Rightarrow U \times C) =$$

$$= \int_{U \times V} 1 \wedge \left(\mu A(u) \to \mu B(v)\right) \wedge \left((1 - \mu A(u) \to \mu A(v)\right)/(u, v),$$

где $\mu A(u) \to \mu A(v) = \begin{cases} 1, \text{ если } \mu A \le \mu B; \\ \mu B, \text{ если } \mu A \ge \mu B. \end{cases}$

Таким образом, возвращаясь к исходной постановке задачи:

если *x* есть *A*, то *y* есть *B*, иначе *y* есть *C*

x есть *A*'

---

y есть *D*,

следствие *D* можно вывести следующим образом:

$Dm = A' \bullet Rm'$,

$Da = A' \bullet Ra'$,

$Db = A' \bullet Rb'$,

$Dgg = A' \bullet Rgg'$.

Пусть имеются следующие посылки:

x – ***не очень маленькое;***

если *x* – ***маленькое****, то y* – ***большое****, иначе y* – ***маленькое.***

Найти значения ***y'***. Множество $U = 1 + 2 + 3$.

маленькое $= {}^{1}/_{1} + {}^{0.4}/_{2}$; ***большое*** $= {}^{0.5}/_{2} + {}^{1}/_{3}$.

Тогда термин ***очень маленькое*** $= {}^{1}/_{1} + {}^{0.16}/_{2}$, а ***не очень маленькое*** $= {}^{0.84}/_{2} + {}^{1}/_{3}$.

Отношение для максиминного правила:

$$Rm' = (A \times B) \cup (\sim A \times C) =$$

$$= \int_{U \times V} \left(\mu A(u) \wedge \mu B(v)\right) \vee \left(1 - \mu A(u)\right) \wedge \mu C(v)\,/(u, v).$$

Здесь A = ***маленькое,*** B = ***большое,*** C = ***маленькое.***

Пример вычисления значений элементов матрицы Rm' приведен ниже:

(u_1, v_1) = (1 ∧ 0) ∨ (0 ∧ 1) = 0; (u_2, v_1) = (0.4 ∧ 0) ∨ (0.6 ∧ 1) = 0.6;

(u_1, v_2) = (1 ∧ 0.5) ∨ (0 ∧ 0.4) = 0.5; (u_2, v_2) = (0.4 ∧ 0.5) ∨ (0.6 ∧ 0.4) = 0.4;

(u_1, v_3) = (1 ∧ 1) ∨ (0 ∧ 0) = 1; (u_2, v_3) = (0.4 ∧ 1) ∨ (0.6 ∧ 0) = 0.4;

(u_3, v_1) = (0 ∧ 0) ∨ (1 ∧ 1) = 1;

(u_3, v_2) = (0 ∧ 0.5) ∨ (1 ∧ 0.4) = 0.4;

(u_3, v_3) = (0 ∧ 1) ∨ (1 ∧ 0) = 0.

$$Rm' = \begin{vmatrix} 0 & 0.5 & 1 \\ 0.6 & 0.4 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix}.$$

Тогда значение y' может быть определено следующим образом

$$y' = \textit{не очень маленькое} \bullet Rm' = \begin{vmatrix} 0 & 0.84 & 1 \end{vmatrix} \bullet \begin{vmatrix} 0 & 0.5 & 1 \\ 0.6 & 0.4 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0.4 & 0.4 \end{vmatrix},$$

т.е. $y' = {}^{1}/_{1} + {}^{0.4}/_{2} + {}^{0.4}/_{3}$, что может быть интерпретировано (с некоторой натяжкой) как ***довольно-таки маленькое***.

Далее рассмотрим для указанных выше посылок арифметическое правило Ra':

$$Ra' = (\sim A \times V + U \times B) \cap (A \times V + U \times C) =$$

$$\int_{U \times V} 1 \wedge \left(1 - \mu A(u) + \mu B(v)\right) \wedge \left((\mu A(u) + \mu C(v)\right)/(u, v) = \begin{vmatrix} 0 & 0{,}5 & 1 \\ 0{,}6 & 0{,}8 & 0{,}4 \\ 1 & 0{,}4 & 0 \end{vmatrix}.$$

Тогда значение y', используя арифметическое правило, может быть определено следующим образом:

$$y' = \textit{не очень маленькое} \bullet Ra' = \begin{vmatrix} 0 & 0.84 & 1 \end{vmatrix} \bullet \begin{vmatrix} 0 & 0.5 & 1 \\ 0.6 & 0.8 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0{,}8 & 0{,}4 \end{vmatrix},$$

т.е. $y' = {}^{1}/_{1} + {}^{0.8}/_{2} + {}^{0.4}/_{3}$.

Вывод с использованием размытого бинарного правила приведен ниже:

$$Rb' = (\sim A \times V \cup U \times B) \cap (A \times V \cup U \times C) =$$

$$= \int_{U \times V} 1 \wedge (1 - \mu A(u) \vee \mu B(v)) \wedge ((\mu A(u) \vee \mu C(v))/(u,v) = \begin{vmatrix} 0 & 0.5 & 1 \\ 0.6 & 0.4 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix}.$$

Тогда значение *y'*, может быть определено следующим образом:

$$y' = \textit{не очень маленькое} \bullet Rb' = \begin{vmatrix} 0 & 0.84 & 1 \end{vmatrix} \bullet \begin{vmatrix} 0 & 0.5 & 1 \\ 0.4 & 0.4 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0.4 & 0.4 \end{vmatrix},$$

т.е. $y' = {}^{1}/_{1} + {}^{0.4}/_{2} + {}^{0.4}/_{3}$, что может быть также интерпретировано как ***довольно-таки маленькое***.

Последний нечеткий вывод проведем с использованием правила Танака-Мидзумото

$$Rgg' = (A \times V \Rightarrow U \times B) \cap (\sim A \times V \Rightarrow U \times C) =$$

$$= \int_{U \times V} (\mu_A(u) \to \mu_B(v)) \wedge ((1 - \mu_A(u)) \to \mu_C(v))/(u,v)$$

где $\mu_A(u) \to \mu_B(v) = \begin{cases} 1, \text{если } \mu_A \le \mu_B \\ \mu_B, \text{если } \mu_A > \mu_B; \end{cases}$;

$$Rgg' = \begin{vmatrix} 0 & 0.5 & 1 \\ 0 & 0.4 & 0 \\ 1 & 0.4 & 0 \end{vmatrix}$$

$$y' = \textit{не очень маленькое} \bullet Rgg' = \begin{vmatrix} 0 & 0.84 & 1 \end{vmatrix} \bullet \begin{vmatrix} 0 & 0.5 & 1 \\ 0 & 0.4 & 0 \\ 1 & 0.4 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0.4 & 0 \end{vmatrix},$$

т.е. $y' = {}^{1}/_{1} + {}^{0.4}/_{2} + {}^{0}/_{3}$, что интерпретируется как ***маленькое***.

Рассмотренные примеры показывают, что использование отношений *Rm'*, *Ra'*, *Rb'* для нечеткого вывода не дают следствий, которые совпадали бы с нашими интуитивными представлениями. Лучшие результаты дает отношение *Rgg'*.

В классической логике известно и еще одно правило вывода, носящее имя *модус толленс*. Записывается оно следующим образом:

$$\begin{array}{l} P \to Q \\ \sim Q \\ \hline \sim P \end{array}$$

В нечетком выводе на основе правила *модус толленс* импликация должна удовлетворять закону контрапозиции, т.е. $P \rightarrow Q = \sim Q \rightarrow \sim P$. Это необходимо для обеспечения эквивалентности правил «если ***x*** есть *A*, то *y* есть *B*» и «если ***y*** есть не ***B***, то ***x*** есть не ***A***».

6.5 Ненадежные знания

В задачах, которые решают экспертные системы, приходится применять ненадежные знания и факты, представить которые двумя значениями – истина и ложь – невозможно. Основополагающим понятием при построении моделей ненадежного вывода является понятие вероятности. Ненадежность иногда представляют байесовской вероятностью; другой подход – это использование коэффициентов уверенности, в задачах с ненадежными данными используется также вероятностная логика [4].

Для решения сложных задач можно использовать метод разбиения их на несколько подзадач. Каждая подзадача, в свою очередь, разбивается также на подзадачи, таким образом, задача в целом описывается иерархически. Для связи между подзадачами с ненадежными знаниями недостаточно операций И и ИЛИ, важную роль играет здесь комбинированная связь, обозначаемая КОМБ. На рисунке 6.2 приведено описание задачи в виде дерева И-ИЛИ-КОМБ.

Если выбрать метод выводов для связей И, ИЛИ, КОМБ, то степень ненадежности можно распространить и на иерархическую сеть выводов. В итоге можно получить степень надежности конечной цели. Существуют общеизвестные методы выбора минимального значения степени надежности из нескольких выводов при связи И и максимального значения при связи ИЛИ, а при связи КОМБ предложен метод MYCIN.

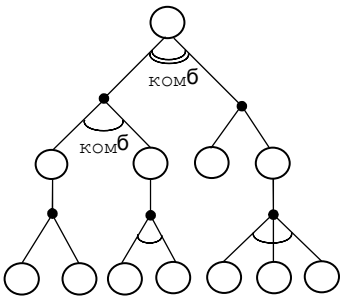


Рис. 6.2

В системе MYCIN имеют дело с ненадежностью, представленной так называемым коэффициентом уверенности CF. Этот коэффициент принимает значения в отрезке [1,–1], когда «1» – заведомо истинно, «–1» – заведомо ложно. Коэффициент уверенности CF(A,(X,Y)) вывода A, если удовлетворяются посылки X и Y, определяется следующим образом:

$$CF\Big(A,(X,Y)\Big)=\begin{cases}1, \text{если } CF(A,X)=1 \text{ или } CF(A,Y)=1;\\ CF(A,X)+CF(A,Y)-CF(A,X)\cdot CF(A,Y), \text{если } CF(A,X)>0 \text{ и } CF(A,Y)>0;\\ CF(A,X)+CF(A,Y), \text{если } CF(A,X)\cdot CF(A,Y)\le 0,\\ \qquad\qquad \text{если } CF(A,X)\ne \pm 1 \text{ и } CF(A,Y)\ne \pm 1;\\ CF(A,X)+CF(A,Y)+CF(A,X)\cdot CF(A,Y), \text{если } CF(A,X)\le 0 \text{ и } CF(A,Y)<0;\\ -1, \text{если } CF(A,X)=-1 \text{ или } CF(A,Y)=-1.\end{cases}$$

Коэффициент уверенности, полученный из трех и более независимых доказательств, можно вывести, последовательно используя указанные выше формулы. При получении положительных и отрицательных значений коэффициента уверенности действует следующее правило: для разнознаковых *CF* сначала применяются формулы второй и четвертой строки, а затем формула третьей строки вычисления *CF*(*A*,(*X*,Y)).

Коэффициент уверенности в системе MYCIN не имеет под собой строгого фундамента, но благодаря простоте восприятия он нашел широкое применение во многих средствах обработки знаний.

Контрольные вопросы

1. Определите формально понятия «далеко», «близко», «рядом».
2. Определите понятия «недалеко», «очень близко», «недалеко и неблизко», выполнив операции над ранее определенными понятиями «далеко», «близко».
3. Определите понятие «нечеткий вывод».
4. Как получаются нечеткие отношения из нечетких условных высказываний?
5. Как может быть использовано обобщенное правило модус толленс для нечетких рассуждений?

7 ОСНОВЫ ПОСТРОЕНИЯ ЭКСПЕРТНЫХ СИСТЕМ

7.1 Структура и разработчики экспертных систем

Экспертная система относится к классу систем, основанных на знаниях, базируется на знаниях эксперта, работает в узких предметных областях и способна объяснить свои действия и результаты. Центральным звеном экспертной системы является мощная база знаний, которые накапливаются в процессе построения системы. *База знаний* разрабатывается как модель ограниченной части мира, которая позволяет, задав механизм вывода, рассуждать об этом мире. Знания представлены в явно сформулированном виде и организованы таким образом, чтобы облегчить процесс принятия решений. Накопление и кодирование знаний является одной из наиболее важных особенностей экспертных систем [7, 8].

Ценность экспертных систем характеризуется следующими аспектами:

- сбором, хранением, распространением экспертных знаний;
- решением проблем, уровень сложности которых соответствует задачам, решаемым специалистами средней квалификации;
- выполнением экспертными системами роли коллективной памяти.

Процесс построения экспертных систем называют *инженерией знаний*. Как правило, он включает в себя некоторый специальный вид взаимодействия между разработчиком экспертной системы, называемым специалистом по технике машинного представления знаний, и одним или несколькими экспертами в некоторой проблемной области. Специалист по технике представления знаний «извлекает» из экспертов их процедуры, подходы и эвристические правила для решения задач и встраивает «извлеченные» знания в экспертную систему.

Итак, основными участниками проектирования экспертных систем являются:

1) *эксперт* – специалист в определенной области знаний, это эрудированный человек, четко выражающий свои мысли, умеющий находить эффективные решения для задач из некоторого определенного поля деятельности;

2) *инженер по знаниям* – специалист по технике машинного представления знаний, это обычно человек с подготовкой по вычислительной технике и ИИ, знающий, как строить экспертные системы; он опрашивает экспертов, упорядочивает знания, решает, как они должны быть представлены в экспертной системе, и может оказывать помощь программистам в написании соответствующего программного кода;

3) *программист* – специалист, имеющий опыт и навыки разработки программных комплексов, знакомый с основными формализмами представления знаний и механизмами выводов в этих формализмах;

4) *конечный пользователь* – человек, использующий экспертную систему после того, как она будет разработана.

Экспертная система представляет собой программный комплекс, который позволяет решать задачи в определенной узкой предметной области. Она называется системой, а не просто программой из-за того, что в ней имеется как компонент решения задачи, так и компонент построения самой системы.

Типичная экспертная система на этапе ее функционирования содержит следующие компоненты:

- *интерфейс* – средство обеспечения общения пользователя с системой;
- *база знаний* – упорядоченный набор сведений об исследуемой предметной области, представленный в некоторой формальной форме (правила, фреймы, семантическая сеть и др.);
- *машина вывода* – формально-логическая система, позволяющая получать новые данные из сведений, размещенных в базе знаний и входных данных;
- *блок объяснений*, позволяющий пользователю убедиться в обоснованности информации, полученной от системы.

Средства построения экспертных систем представляют собой системы программирования, облегчающие работу по созданию экспертных систем. Средства эти можно разделить на четыре основные категории:

1) языки программирования (проблемно-ориентированные, языки для обработки символов);

2) языки инженерии знаний (оболочки, языки общего назначения);

3) вспомогательные средства построения систем (средства накопления знаний, средства проектирования);

4) обеспечивающие средства (средства программирования, средства объяснения).

7.2 Основные функции экспертных систем

Разработанные экспертные системы предназначены для решения самых разнообразных задач, но их основные функции могут быть сгруппированы по своему роду в категории [7].

Интерпретация – получение описания ситуации на основе наблюдаемых данных. Примером может служить интерпретация показа-

ний измерительных приборов на предприятии химической промышленности с целью получения выводов о состоянии процесса. В системах интерпретации непосредственно используются реальные данные, а не идеальные символьные представления проблемной ситуации. В них приходится сталкиваться с такими задачами, которые не решаются в системах многих других типов. Это объясняется тем, что в системах интерпретации необходимой может оказаться обработка данных, которые подвержены воздействию шума, немногочисленны, неполны, ненадежны или ошибочны. В них требуется использование специальных методов, направленных на извлечение характеристик из непрерывных потоков данных, сигналов или картин, и способов их символьного отображения. В системах интерпретации возможна обработка данных различных видов. Например, в системах распознавания как зрительных образов, так и речи используются естественные входные данные (визуальные изображения в одном случае и звуковые сигналы в другом) для получения характеристик и смыслового значения. В химических системах интерпретации используются данные о дифракции рентгеновских лучей или масс-спектральные данные и данные ядерного магнитного резонанса для вывода структуры соединений. В медицинских системах интерпретаций используются измерения характеристик пациента (например, пульс, кровяное давление) с целью диагностики и лечения заболеваний. В системах интерпретации военного назначения используются сигналы радиолокационных станций, радио- и звуковых устройств для оценки ситуаций и идентификации целей.

Прогнозирование – получение вероятных последствий из заданных ситуаций. Примерами могут служить:

1) прогнозирование ущерба, наносимого урожаю некоторой разновидностью насекомых;

2) получение оценки глобального спроса на нефть на основе сложившейся геополитической ситуации в мире;

3) предсказание того места в мире, в котором, вероятно, произойдет очередной вооруженный конфликт, на основе данных разведки.

В системах прогнозирования на основе конкретных входных данных иногда используются имитационные модели для генерации ситуаций или сценариев, которые могут возникнуть. Имитационные модели – это программы, отражающие происходящее в реальном мире. Эти потенциальные ситуации вместе со знаниями о порождающих их процессах образуют базу для получения прогнозов. На сегодняшний день специалистами по ИИ разработано сравнительно мало систем прогнозирования, возможно, в силу сложности разработки имитационных моделей и обеспечения взаимодействия с ними.

Ниже приводится пример правила из системы прогнозирования. Система PLANT/cd использует имитационную модель для выполнения необходимых расчетов и предназначена для прогнозирования ущерба, наносимого урожаю гусеницей озимой совки.

ЕСЛИ

1) рассчитана таблица зависимости развития гусеницы совки от фазы листа;

2) известно, имеется ли в ряду свыше 13 семян на метр;

3) известна разновидность зерна;

4) известна влажность почвы на поле,

ТО вычислить урожай зерна без учета обработки инсектицидами и присвоить полученное значение переменной YIELD1.

Диагностика – выявление неисправностей системы по наблюдениям. Примерами могут служить системы определения причины заболеваний по наблюдаемым симптомам, локализации отказов в электрических цепях и обнаружения неисправных блоков в системах охлаждения ядерных реакторов. Системы диагностики нередко выступают в роли консультантов, осуществляющих не только диагностику в случае затруднений, но и оказывающих помощь при доводке. Они могут взаимодействовать с пользователем с целью оказания помощи в обнаружении неполадок и после этого предлагать рекомендуемые меры по их устранению. Область медицины представляется вполне естественной для применения диагностики, и, действительно, для медицины было разработано диагностических систем больше, чем для любой другой отдельной проблемной области. Однако в настоящее время разрабатывается большое число диагностических систем, предназначенных для использования в технических и вычислительных системах.

Ниже приводится правило для диагностической системы MYCIN, предназначенной для диагностики и лечения инфекционных болезней.

ЕСЛИ

1) организм относится к числу грамположительных;

2) морфологически организм представлен кокковой флорой;

3) форма роста организма имеет вид цепочек,

ТО со степенью уверенности (0,7) можно предположить, что организм тождественен стрептококку.

Проектирование – выбор конфигурации объектов в условиях ограничений. Примерами могут служить системы конструирования плат интегральных схем и синтеза сложных органических молекул. В системах проектирования нередко используется синтез для конструирования частичных проектов и имитации для проверки и испытания проектных решений. Из-за тесной взаимосвязи проектирования с планированием во многих системах проектирования предусматриваются механизмы, предназначенные для разработки и отладки планов с целью

получения требуемого построения. Системы проектирования могут существенно экономить затраты на необязательный поиск в результате создания планов, предназначенных для получения требуемой конфигурации, и оценки их в условиях налагаемых задачей ограничений. Двумя наиболее популярными областями применения систем построения являются молекулярная биология и микроэлектроника.

Планирование – составление плана действий. Примерами таких систем могут служить системы разработки плана по применению ряда химических реакций к группам атомов с целью получения сложного органического соединения или создания плана воздушного налета, разрабатываемого в течение нескольких дней, для снижения боеспособности вооруженных сил противника. Планирующие системы нередко должны отвергать некоторую конкретную линию поведения или часть плана из-за того, что она не укладывается в рамки ограничений задачи, и переходить к более ранней точке или ситуации, из которой анализ должен начинаться заново. Такие отступления могут дорого обходиться, и поэтому в некоторых системах планирования решаемая задача разбивается на ряд подзадач, которые стремятся упорядочить таким образом, чтобы избежать повторного планирования из неудачно выбранных точек. Наиболее частой сферой приложения для систем планирования служат химия, электроника, военное дело.

Ниже приводится пример правила для системы планирования. Системой TATR осуществляется планирование воздушных налетов на аэродромы противника. Это правило облегчает получение оценки о целесообразности бомбардировки целей на аэродроме противника.

ЕСЛИ

1) на аэродроме обнаружены самолеты;

2) число самолетов, расположенных на аэродроме без укрытия, превышает четверть от общего числа летных средств на этом аэродроме,

ТО оценка целесообразности бомбардировки аэродрома равна «превосходно».

Мониторинг – сравнение наблюдений с критическими точками плана. Примером могут служить системы контроля за показаниями аппаратуры в ядерных реакторах. Системами мониторинга фиксируется наблюдаемое поведение, подтверждающее их «представления» о нормальном поведении или их предположения о возможных отклонениях в поведении. Системы мониторинга по самой своей природе имеют дело со временем и должны осуществлять зависящую как от обстановки (контекста), так и от времени интерпретацию фиксируемого ими поведения. Это может означать необходимость запоминания всех значений, которые принимал некоторой параметр в системе в различные

промежутки времени, так как скорость и знак изменения могут иметь не менее важное значение, чем само действительное значение рассматриваемого параметра в любой момент времени.

Ниже приводится пример правила из системы мониторинга. Система REACTOR предназначена для наблюдения за показаниями аппаратуры в ядерном реакторе.

ЕСЛИ

1) наблюдается неправильный теплоотвод от системы первичного охлаждения в систему вторичного охлаждения;

2) слабый поток водоснабжения,

ТО неисправность заключается в выходе из строя системы водоснабжения.

Обучение – диагностика ошибок при изучении курса или дисциплины и подсказка правильного решения.

Управление – поддержание определенного режима деятельности.

7.3 Этапы разработки экспертных систем

Процедура получения знаний у специалиста и занесения их в программу называется *извлечением знаний*. Разработка экспертной системы, по сути, основывается на передаче опыта решения задач с последующим его преобразованием во внутреннее представление. Практически процесс разработки экспертной системы для любой предметной области можно разделить на следующие этапы.

Этап идентификации. На этом этапе решаются следующие задачи:

- определяются цели, проблемы, задачи;
- находится эксперт (эксперты) и остальные разработчики системы;
- определяются ресурсы.

Работа начинается с выбора проблемы. Проблема не должна быть ни слишком легкой (ее решение не должно занимать у эксперта несколько минут), ни слишком трудной (на ее решение эксперт не должен тратить несколько месяцев). Перечислим базовые критерии для выбора проблемы:

- символьные рассуждения; нет смысла разрабатывать экспертную систему для численных расчетов;
- наличие экспертов, возможность их сотрудничества и общая согласованность мнений;
- важность проблемы; это могут быть как проблемы, требующие высокого уровня экспертизы, так и проблемы невысокой сложности, но часто повторяемые и трудные для проверки;

• круг решаемых задач; должна иметься некоторая спецификация для предполагаемых функций экспертной системы и характеристик решаемых ею задач;

• наличие исходных и тестовых данных;

• возможность постепенного наращивания системы.

Определение проблемы заключается в составлении неформального описания решаемой проблемы. Данное описание должно включать следующие пункты:

• общую характеристику проблемы;

• разбиение проблемы на подпроблемы;

• ключевые понятия и отношения;

• входные данные;

• предположительный вид решения;

• необходимые для решения проблемы знания;

• прецеденты решения рассматриваемой проблемы.

Цели также формулируются в явном виде. Примерами возможных целей могут быть следующие формулировки: автоматизация рутинных действий пользователя; формализация знаний эксперта; улучшение качества решений; распространение опыта и др.

Задача определения участников сводится к определению количества участников разработки и установлению формы их взаимоотношений.

Типичными ресурсами при создании ЭС являются источники знаний (дополнительные эксперты, книги, методики, отчеты), время разработки, вычислительные средства, финансовые расходы.

Этап концептуализации. На этом этапе эксперт и инженер по знаниям выявляют основные понятия, отношения и характер информационных потоков, которые необходимы для описания процесса решения задач в данной предметной области.

На этапе концептуализации выявляется структура знаний и определяются следующие характеристики:

- ❑ терминология; список основных понятий и их атрибуты;
- ❑ взаимосвязи между понятиями и типы отношений (род-вид, часть-целое, причина-следствие и др.);
- ❑ входные и выходные данные;
- ❑ используемые стратегии и гипотезы;
- ❑ ограничения, накладываемые на стратегии решения.

Для определения перечисленных характеристик целесообразно составить протокол действий и рассуждений эксперта в процессе решения одной конкретной задачи.

Этап формализации. Этот этап связан с переводом основных понятий и отношений в формальное представление, выбранное инжене-

ром по знаниям. В качестве такого формализма выступает язык представления знаний.

Этап реализации. Система-прототип, решающая поставленные задачи и включающая в себя базу знаний и механизм вывода, является результатом данного этапа. Создание прототипа состоит в оригинальной разработке отдельных его компонент или выборе их из существующих инструментальных средств, а также наполнения базы знаний. Задача данного этапа – разработка программного комплекса, подтверждающего жизнеспособность выбранных методов решений и способов представления. При разработке первого прототипа обычно не рассматриваются вопросы, связанные с анализом и синтезом фраз на естественном языке, моделирование рассуждений с нечеткими понятиями, учет сложных временных, причинных и модальных отношений. В данном прототипе реализуется простейшая процедура вывода, здесь важно получение решения, но не эффективность его. Повышение эффективности работы системы – задача, которую решает следующий прототип экспертной системы. Выполнение экспериментов с первым прототипом, анализ пожеланий и замечаний служат основой для создания второго прототипа. При разработке второго прототипа, кроме указанных выше, решаются следующие задачи: анализ работы системы при расширении базы знаний; исследование возможностей системы при решении более широкого круга задач; анализ мнений и пожеланий пользователей о работе системы; разработка системы ввода/вывода на ограниченном естественном языке.

Этап тестирования. На этом этапе производится оценка работы прототипа с целью приведения в соответствие с реальными запросами пользователей. Здесь проверяются: управляющие стратегии; полнота и непротиворечивость базы знаний; удобство и адекватность системы ввода/вывода; качество самих тестовых примеров.

Перечисленные этапы создания экспертной системы итеративны, не являются четко очерченными, детально определенными и независимыми. Они лишь приблизительно описывают сложный процесс, названный извлечением знаний.

7.4 Стадии разработки системы

Стадии характеризуют степень проработки и отлаженности экспертной системы. Принято выделять следующие стадии существования экспертных систем:

- демонстрационный прототип;
- исследовательский прототип;
- действующий прототип;

- промышленная система;
- коммерческая система.

Демонстрационным прототипом называют систему, решающую часть задач, демонстрирующую жизнеспособность принятого подхода. Прототип работает, имея несколько десятков правил или понятий.

Исследовательским прототипом называется система, которая решает большинство задач, но не устойчива в работе и не полностью проверена. Данный прототип имеет в базе знаний несколько сотен правил.

Действующий прототип – это система, надежно решающая все задачи, но для решения сложных задач требующая много времени и памяти. Данный прототип имеет в базе знаний до тысячи правил.

Промышленная система обеспечивает высокое качество решений при минимуме времени и памяти. Данная система может содержать более тысячи правил и переписана с использованием более эффективных средств представления, например, система перекодируется на язык более низкого уровня. Хотя эта операция не является обязательной и выполняется для тех систем, в которых знания о предметной области не будут меняться в ближайшем будущем. Системы, в которых предметная область меняется, поддерживаются инструментальными средами и не перекодируются.

Коммерческая система пригодна не только для собственного использования, но и для продажи, то есть она хорошо документирована, обладает дружественным интерфейсом и снабжена надлежащим сервисом.

7.5 Инструментальные средства разработки

Стиль разработки интеллектуальных систем существенно отличается от стиля разработки обычных компьютерных систем. В табл. 7.1 приведены некоторые характерные отличия между обычными и интеллектуальными системами.

Таблица 7.1

Характеристика	Программирование систем	
	Интеллектуальных	Традиционных
Тип обработки	Символьный	Числовой
Методы	Эвристический поиск	Алгоритм
Задание шагов решения	Неявное	Явное
Знания	Неточные	Точные
Модификации	Частые	Редкие

Инструментальные средства построения экспертных систем представляют собой системы программирования, облегчающие работу по созданию экспертных систем. Различают следующие типы инструментальных средств разработки:

- языки программирования;
- языки представления знаний;
- средства автоматизации проектирования;
- оболочки.

Язык программирования – это механизм абстрагирования. Он дает возможность программисту описать вычисления абстрактно и в то же время позволяет компилятору перевести это описание в детализированную форму, необходимую для выполнения на компьютере. Языки программирования существуют только для преодоления разрыва в уровне абстракции между аппаратными средствами и реальным миром. Чтобы выбрать язык программирования, следует избрать соответствующий уровень абстракции; и нет ничего удивительного в том, что какой-либо язык может подходить для одного проекта и не подходить для другого.

Языки программирования, применяемые в экспертных системах, как правило, относятся к числу либо проблемно-ориентированных (типа Фортрана или Паскаля), либо языков манипулирования символами (типа Лиспа или Пролога). Проблемно-ориентированные языки предназначены для решения проблем определенных классов. Например, Фортран характеризуется удобными возможностями для выполнения алгебраических действий и наиболее приемлем для решения задач математического и статистического характера. Языки манипулирования над символами предназначены для решения прикладных задач в области искусственного интеллекта. В основу языка Лисп положены следующие принципы: использование единого спискового представления для программ и данных; применение выражений для определения функций; скобочный синтаксис языка. Иная модель организации вычислений положена в основу языка Пролог. Здесь программист описывает предметную область, задавая для этого необходимые факты и правила, а ПРОЛОГ-система сама строит дедуктивный вывод на введенных данных. Логическая программа представляет собой выполнимую формулу в некоторой логике.

Языки, реализующие ту или иную модель представления знаний, называют *языками представления знаний*. Напомню, что традиционно выделяют четыре модели представления знаний – логическую, продукционную, фреймовую, сетевую, и соответственно можно говорить о четырех типах языков представления знаний. Наибольшее предста-

вительство имеют фреймовые языки: GUS, KRL, FRL, OWL и др. Однако самым распространенным языком представления знаний стал продукционный язык OPS5, краткое знакомство с которым дано в разделе описания продукционных моделей.

Средства автоматизации проектирования предоставляют разработчику разнообразные средства для учета особенности приложений. Такие средства объединяют в рамках одной среды различные методы решения задач, представления знаний и вывода. Лучшими представителями этого класса инструментальных средств являются ART и KEE.

Интегрированная среда ART объединяет два формализма представления знаний: правила и фреймы, причем приоритет здесь отдается правилам. Модели вывода в среде ART традиционны: прямой и обратный, однако они могут объединяться в один механизм вывода. ART предлагает дружественный интерфейс с развитой системой подсказок, системой меню и графическим пакетом, а также редактор баз знаний.

В отличие от ART, инструментальное средство KEE является средой, в основе которой лежат фреймы. Причем здесь слоты фреймов могут быть связаны с процедурными знаниями. Описание объектов и правил в KEE представляется в виде иерархии фреймов. Каждый объект представляется слотами, которые могут иметь различные аспекты; последние, в свою очередь, могут иметь множественные значения. Интерфейс включает редактор базы знаний и графические средства.

Оболочки содержат все компоненты экспертной системы в готовом виде. Оболочки представляют собой экспертную систему с удаленными из нее специфичными для соответствующей области знаниями, в которой оставлены машина вывода и обеспечивающие средства. Оболочки предоставляют структурные и встроенные средства, превращающие разработку экспертной системы в простой и быстрый процесс. Однако им не хватает общности и гибкости; их применение ограничивается лишь некоторым классом задач, и они значительно снижают предоставляемые разработчику возможности по построению экспертных систем. Ниже рассмотрим оболочку EMYCIN.

EMYCIN – это предметно-независимая версия системы MYCIN. EMYCIN является инструментом для разработки консультационной программы, которая может запрашивать данные о положении дел и обеспечивает их интерпретацию или анализ. Она удобна для решения дедуктивных задач, таких как диагностика заболеваний или неисправностей, для которых характерно большое количество ненадежных входных измерений, а пространство решений может быть четко очерчено.

EMYCIN позволяет применить дедуктивную машину системы MYCIN к предметной области новой задачи, специальные знания кото-

рой могут быть представлены на языке правил системы MYCIN. При этом EMYCIN сохраняет все особенности системы MYCIN, а также программное обеспечение построения системы, что облегчает отладку и ввод данных в базу знаний. EMYCIN использует следующий язык правил:

<правило> ::= (ЕСЛИ <антецедент> ТО< действие > {ИНАЧЕ < действие >})
<антецедент> ::= ({И {< условие >}$^{+}$)
<условие> ::= (ИЛИ {< условие >}$^{+}$) | (<предикат> < ассоциативная_тройка >)
<ассоциативная_тройка>::= (< атрибут > <объект> < значение >)
< действие > ::= {< консеквент >}$^{+}$ | {< процедура >}$^{+}$
< консеквент > ::= (< ассоциативная_ тройка > < фактор_достоверности >)

Правило связывает антецедент с одним действием, если значением его антецедента является истина, или с другим действием, если значением его антецедента является ложь. Антецедент – это конъюнкция одного или более условий. Условие представляет собой либо дизъюнкцию одного или более условий, либо предикат, примененный к тройке атрибут-объект-значение. Поскольку предикат может включать отрицание, антецедент можно рассматривать как произвольную булевскую комбинацию предикатов над ассоциативными тройками.

Например, антецедент одного из правил бактериальной инфекции в системе MYCIN выглядит следующим образом [8]:

```
(И
  (ПОДОЗРЕВАЕТСЯ (ТОЖДЕСТВЕННОСТЬ ОРГАНИЗМ
                          CORYNEBACTERIUM_NON-DIPHTHERIAE))
  (ИЛИ
     (НЕ-ПОДОЗРЕВАЕТСЯ (ПОСТОРОННИЙ ОГРАНИЗМ ИСТИНА))
     (ПОДОЗРЕВАЕТСЯ (СУЩЕСТВЕННЫЙ ОГРАНИЗМ ИСТИНА))
  )
)
```

На естественном языке это означает: антецедент принимает значение «истина» тогда и только тогда, когда подозревается, что рассматриваемый организм принадлежит к определенному типу (CORYNEBACTERIUM NON-DIPHTHERIAE) и либо не рассматривается как посторонний (несущественный), либо считается связанным с какой-то серьезной болезнью.

Объекты в ассоциативных тройках в терминологии EMYCIN называются «контекстами». Объектами являются переменные, соответствующие объектам предметной области. Они организованы в иерархию, которая называется контекстным деревом. Например, объектами могут быть ПАЦИЕНТ-1, КУЛЬТУРА-1, ОРГАНИЗМ-1, ОРГАНИЗМ-2, а контекстное дерево будет показывать, что ОРГАНИЗМы принадлежат КУЛЬТУРАм, а КУЛЬТУРы принадлежат ПАЦИЕНТам. Контекстное дерево обеспечивает некоторый механизм наследования, присущий представлениям в виде фреймов. Например, поскольку культуры имеют местоположение, система, зная, что ОРГАНИЗМ-2 произошел из

КУЛЬТУРы-1, и найдя местоположение КУЛЬТУРы-1, может определить местоположение ОРГАНИЗМа-2.

Для того чтобы включить ненадежные данные, система EMYCIN связывает с каждой тройкой *атрибут-объект-значение* фактор достоверности. Это число представляет собой ненормированную вероятность, принимающую значения из отрезка от -1, когда ассоциативная тройка имеет значение «ложь», до +1, когда ассоциативная тройка имеет значение «истина». Здесь 0 означает отсутствие какого-либо мнения на этот счет. Предикаты, такие как ПОДОЗРЕВАЕТСЯ, могут принимать значение «истина» для некоторого доверительного интервала (от 0.2 до 1) либо быть функциями, показывающими степень истинности. Предикат И возвращает минимальное значение, предикат ИЛИ возвращает максимальное значение достоверности своих аргументов. Однако в качестве условия применения правила антецедент считается «истинным», если результирующая достоверность больше некоторого порогового значения (обычно 0.2), и считается «ложным», если его результирующая достоверность меньше некоторого другого порогового значения (обычно –0.2).

Действия правила заключаются либо в уточнении достоверности указанных консеквентов, либо в выполнении соответствующего набора присоединенных процедур. Такое уточнение достоверности консеквента означает изменение достоверности того, что атрибут этого объекта имеет конкретное значение в свете новой информации, идущей от антецедента. Альтернативное действие, заключающееся в обращении к присоединенным процедурам, представляет собой механизм выхода из ситуации и перехода к выполнению произвольного лисповского кода.

Основная стратегия управления, используемая системой EMYCIN, – это обратная цепочка рассуждений. EMYCIN пытается последовательно применять правила до тех пор, пока она либо установит искомое значение, либо исчерпает список правил. Если никакого значения вывести нельзя, то EMYCIN обращается к пользователю, запрашивая это значение.

При попытке применить правило система EMYCIN должна сначала установить истинность его антецедента. Чтобы сделать это, система обычно должна установить значения атрибутов объектов. Это приводит к формулировке подцелей, обращение к которым производится посредством рекурсивного использования того же самого механизма. Таким образом, применение одного правила устанавливает подцели, которые, в свою очередь, активизируют другие правила, а попытка достижения целей направляет весь процесс консультации.

7.6 Средства объяснения

Важность объяснений в экспертных системах вызвана рядом факторов. Во-первых, трудно ожидать, что пользователи будут знать все возможности и понимать все действия экспертной системы. В связи с этим пользователю требуется помощь экспертной системы. Во-вторых, значимость объяснений обусловлена тем, что экспертные системы предназначены для использования в слабо формализованных областях, т.е. для решения задач, не имеющих алгоритмических решений. В условиях отсутствия теории, являющейся надежной гарантией правильности полученных результатов, возникает особая необходимость в разработке средств, дающих пользователям возможность убедиться в достоверности методов и знаний, используемых экспертной системой для получения решения. В качестве таких средств в экспертных системах используются объяснительные способности. Объяснительные способности экспертных систем должны быть ориентированы на всех, кто с ними взаимодействует. В нашем курсе такие люди обозначены обобщенным термином «пользователи». Обычно выделяют следующие типы пользователей:

1) пользователи, не являющиеся специалистами в области экспертизы; их задача – получить от экспертных система решение некоторой задачи;

2) пользователи, являющиеся специалистами в области экспертизы, их задача, используя экспертные системы, сократить трудоемкость получения результата или повысить его качество;

3) «студенты», т.е. пользователи, которые с помощью экспертных систем хотят обучиться методам решения задач из области экспертизы;

4) эксперты, т.е. высококвалифицированные специалисты, в задачу которых входит обнаружение недостающих знаний и ввод их в систему;

5) инженеры по знаниям, т.е. специалисты в области инженерии знаний, в задачу которых входит отладка управляющего механизма, анализ и модификация экспертной системы.

Специфика задач, решаемых пользователями различных типов, предъявляет к объяснительным способностям экспертных систем различные требования. Так, основная цель использования объяснительных способностей для «студента» – обучение, для эксперта и инженера по знаниям – локализация ошибок, для пользователя-специалиста – обеспечение доверия к результату, для пользователя-неспециалиста – достижение взаимопонимания. Таким образом, можно выделить сле-

дующие цели, преследуемые при использовании объяснений в экспертных системах:

1) локализовать ошибки системы путем исследования метода рассуждения;

2) повысить доверие пользователя к системе путем объяснения способа получения результата;

3) достичь взаимопонимания между пользователем и системой путем объяснения непонятных терминов, ответов и ситуаций;

4) обучить пользователя.

Взглянем на проблему объяснения несколько шире. В научной литературе обычно выделяют пять основных типов объяснения:

1) причинные (каузальные);
2) объяснения через закон;
3) функциональные (целевые, мотивационные);
4) структурные;
5) генетические (исторические).

Причинные объяснения вскрывают причинные взаимосвязи между некоторыми явлениями. Объяснить в этом случае – значит вскрыть причину. Важность причинных объяснений вытекает из того, что причинность всеобща, т.к. нет явлений, которые не имели бы своих причин, как нет явлений, которые не порождали бы тех или иных следствий.

Наиболее развитой формой научного объяснения является объяснение *на основе теоретических законов*. Объяснение через закон сводится к установлению, в соответствии с каким законом, теорией, моделью возникло или происходило объясняемое явление. В этом случае объяснение можно рассматривать как логическую операцию дедукции, т.е. выведение частных следствий из общего закона (теории, модели).

Функциональное объяснение сводится к установлению функций, выполняемых той или иной частью системы. Эти объяснения строятся по принципу: «*X* нужно для того, чтобы могло произойти *Y*». Например, «мимикрия нужна для того, чтобы скрываться от врагов». Одним и тем же явлениям можно давать и причинные объяснения и функциональные.

Структурное объяснение дается посредством описания структуры, которая обеспечивает выполнение функций и поведение объясняемой системы в целом. В отличие от других типов объяснений, использующих преимущественно какие-то отдельные аспекты явлений (причины, цели и т.д.), структурное объяснение призвано воспроизвести ситуацию в целом.

Генетическое или историческое объяснение состоит в раскрытии условий, причин и законов, приведших к текущему состоянию систе-

мы, предмета, явления. Генетическое объяснение вскрывает происхождение сущности и способ ее образования.

В существующих экспертных системах, как правило, используются целевые и причинные объяснения. При практической реализации в функции объяснительного компонента экспертной системы обычно включают генерацию любых ответов на запросы пользователей. Некоторые из этих ответов могут быть отнесены к объяснениям только условно. Так, ответы на вопросы КАКОЙ, СКОЛЬКО, ЧТО, КТО и т.п. в отличие от ответов на вопросы ПОЧЕМУ и КАК, не являются объяснениями, они просто сообщают запрашиваемое значение или оценивают истинность некоторого факта. Выделяют следующие подходы к созданию программ, обладающих объяснительными способностями:

1) записанные объяснения, т.е. использование для объяснений заранее подготовленных текстов на естественном языке;

2) генерация объяснений непосредственно из программных кодов.

Простейший способ получения объяснений о том, что делает программа, состоит в запоминании ответов на все возможные вопросы. Данный подход применим в том случае, если можно предвидеть все вопросы. При этом подходе пользователь получает объяснение точно в том виде, в котором оно было записано. Типичными примерами записанных объяснений являются сообщения о стандартных ошибках, обнаруживаемых транслятором языка программирования. Достоинства и недостатки указанного подхода очевидны. Данный подход применялся в ранних экспертных системах (например, в DENDRAL) или в экспертных системах, где специфика проблемной области позволяла стандартизовать взаимодействия с пользователями (например, в системе R1).

В случае генерации объяснений непосредственно из программных кодов объяснительный компонент исследует текст выполняемой программы и на его основе генерирует объяснение. Типичным примером экспертной системы, использующей такой подход, является MYCIN. Здесь объяснительный компонент обрабатывает три вида вопросов:

1) вопросы о действиях системы;

2) вопросы о динамических знаниях, т.е. о знаниях, используемых системой в ходе решения задачи;

3) вопросы о статических знаниях, т.е. о знаниях, хранимых в базе знаний.

Система MYCIN в состав объяснительного компонента включает дерево целей. Оно содержит следующее:

1) частично упорядоченное множество целей;

2) информацию о способе достижения каждой цели, т.е. выведена ли цель с помощью правила или сообщена пользователем;

3) информацию о результате, т.е. успехе или достижении цели либо неудачи или недостижении цели.

В ходе решения задачи экспертная система может задавать пользователю вопросы. Если вопрос покажется пользователю неуместным, то он может прервать работу системы и обратиться к объяснительным средствам системы, чтобы выяснить, с какой целью система задала этот вопрос. В связи с тем, что объяснения рассматриваются в терминах движения от текущей цели по дереву целей, пользователю предоставлены две основные команды: ПОЧЕМУ и КАК. Ответ на вопрос ПОЧЕМУ соответствует движению вверх по дереву в поисках ближайшей цели, объясняющей «почему» достигается текущая цель. Ответ на вопрос КАК соответствует движению вниз по дереву целей и объясняет, как достигалась или будет достигаться текущая цель. Недостатки:

1) ограничения, состоящие в том, что вопросы ПОЧЕМУ и КАК интерпретируются только в терминах правил и целей.

2) не учитывает адресата (тип пользователя).

7.7 Приобретение знаний

Процесс получения знаний от эксперта или из других источников и передача их экспертной системе называют приобретением знаний. Обычно источником знаний является эксперт-человек, но могут быть и эмпирические данные и тексты, в которых содержатся сведения об области экспертизы. Сложность данного процесса обусловлена в основном двумя факторами: большим объемом знаний, используемых экспертом; знания эти не полностью осознаются экспертом.

Получить знания о предметной области можно различными способами. Перечислим некоторые из них:

- извлечение знаний из книг, справочников, отчетов, инструкций, документов;
- свободное описание предметной области экспертом;
- интервьюирование эксперта;
- получение знаний от эксперта, выступающего в роли лектора;
- получение знаний в режиме мозгового штурма.

Процесс приобретения знаний можно рассматривать с различных точек зрения и характеризовать множеством аспектов. Мы рассмотрим два аспекта: фазы приобретения знаний, модели приобретения знаний.

ФАЗЫ ПРИОБРЕТЕНИЯ ЗНАНИЙ. Наличие нескольких фаз в процессе приобретения знаний признается всеми специалистами. Расхож-

дения существуют в вопросе о том, сколько имеется фаз и какие они. Ряд авторов считает, что в процессе приобретения знаний есть столько же фаз, сколько в процессе проектирования экспертной системы, а именно: идентификация, концептуализация, формализация, выполнение и тестирование. Другие авторы выделяют следующие фазы: определение предметной области, формирование начальных фундаментальных знаний, формирование основного состава знаний.

Рассмотрим три фазы приобретения знаний: *предварительную*, *начальную*, *фазу накопления*. Предварительная фаза приобретения знаний характеризуется тем, что экспертной системы еще не существует. Знания приобретаются инженером по знаниям от эксперта. На этой фазе задача инженера по знаниям состоит в том, чтобы получить от эксперта основные сведения об области экспертизы, сюда входят, например, основные понятия, отношения, подзадачи. Далее на их основе формируется общее представление о структуре и принципах построения экспертной системы.

На начальной фазе осуществляется наполнение системы знаниями о представлении, т.е. значениями, определяющими организацию, структуру и способ представления базы знаний. Для определения указанных знаний необходимо владеть основами программирования и детально понимать функционирование проектируемой экспертной системы. В связи с этим, введение знаний на начальной фазе может осуществлять только инженер по знаниям, а не эксперт.

В ходе фазы накопления осуществляется приобретение основных знаний об области экспертизы. Приобретение знаний на этой фазе ведется экспертом совместно с инженером по знаниям. Здесь решаются следующие задачи:

1) обнаружение неправильности, неполноты или противоречивости знаний, используемых экспертной системой;

2) извлечение новых знаний, устраняющих обнаруженную неправильность, неполноту или противоречивость;

3) преобразование новых знаний в вид, понятный экспертной системе;

4) объединение «новых» знаний со «старыми».

Можно говорить о приобретении знаний в узком и широком смысле. В узком смысле под приобретением знаний понимают фазу накопления, когда происходит передача знаний в действующую экспертную систему. В широком смысле под приобретением знаний понимают все фазы. По-разному интерпретируется понятие «передача знаний экспертной системе» в разных трактовках. В первом случае под экспертной системой понимается действующий прототип системы, которому передаются знания, а во втором случае знания передаются как

экспертной системе, так и инженеру по знаниям, который их воплотит в виде действующего прототипа экспертной системы.

МОДЕЛИ ПРИОБРЕТЕНИЯ ЗНАНИЙ. Процесс приобретения знаний является наиболее сложным этапом разработки экспертной системы. Это объясняется тем, что обычно инженер по знаниям плохо разбирается в предметной области, а эксперт не знает программирования. В связи с этим лексика, используемая экспертом, непонятна инженеру по знаниям. Чтобы уточнить и расширить лексику, требуется совместная работа эксперта и инженера по знаниям. Одна из наиболее сложных задач, стоящих перед инженером по знаниям, заключается в том, чтобы помочь эксперту структурировать знания по предметной области. Процесс приобретения знаний можно свести к последовательности выполнения следующих задач:

1) определяется необходимость модификации знаний;

2) при необходимости модификации осуществляется извлечение новых знаний, в противном случае процесс приобретения знаний заканчивается;

3) новые знания преобразуются в форму, понятную экспертной системе;

4) знания системы модифицируются, и осуществляется переход к первой задаче.

В выполнении перечисленных задач могут принимать участие эксперт, инженер по знаниям, программист и экспертная система. В зависимости от того, кто выполняет задачу, можно выделить различные модели приобретения знаний.

В ранних работах по искусственному интеллекту взаимодействие с работающей системой осуществлял только программист. При разработке системы программисты не отделяли знания (данные) от механизма вывода. В задачу программиста входило освоить с помощью эксперта предметную область и затем при разработке системы выступать в роли эксперта и программиста. Недостаточное знание области экспертизы не позволяло программисту гарантировать полноту и непротиворечивость приобретенных знаний. Кроме того, неизбежные модификации системы приводили к невозможности сохранить однажды достигнутую непротиворечивость знаний из-за отсутствия разделения системы на базу знаний и механизм вывода. Модель такого взаимодействия приведена на рис. 7.1. Здесь все перечисленные выше задачи по приобретению знаний выполнял программист.

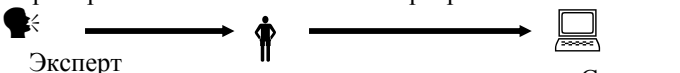


Рис. 7.1

Последующие разработки систем искусственного интеллекта основывались на отделении знаний от программ и оформлении знаний в виде простых информационных структур, называемых базами знаний. В этом случае эксперт взаимодействует с системой либо непосредственно, либо через инженера по знаниям (рис. 7.2).

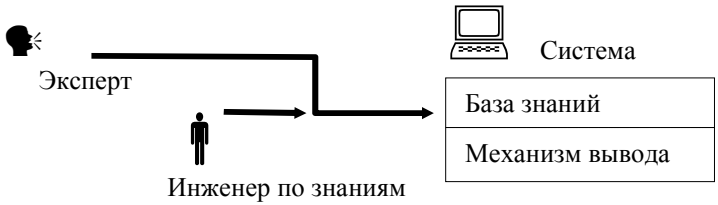


Рис. 7.2

Преимущество данного подхода по сравнению с предыдущей моделью состоит в том, что база знаний упрощает модификацию знаний. В данной модели первую и вторую задачи приобретения знаний выполняет эксперт с помощью инженера по знаниям, третью задачу выполняет инженер по знаниям, а четвертую – экспертная система. Недостатком данного подхода является его большая трудоемкость, т.к. из четырех задач приобретения знаний автоматизирована только одна.

Эксперт, минимально сведущий в вопросах программирования, может взаимодействовать с экспертной системой через интеллектуальный редактор без посредничества инженера по знаниям. Модель этого подхода изображена на рис. 7.3.

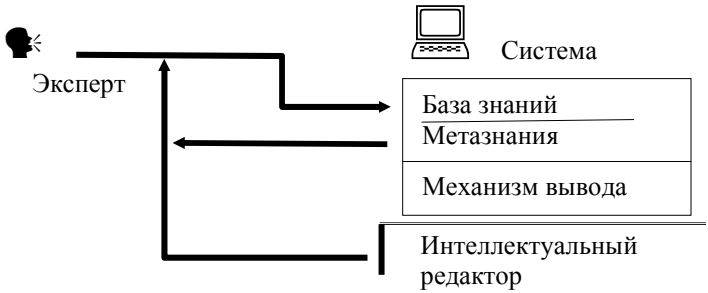


Рис. 7.3

В этой модели интеллектуальный редактор должен обладать развитыми диалоговыми способностями и значительными знаниями о структуре базы знаний, т.е. метазнаниями. Интеллектуальный редактор может быть включен в состав экспертной системы. При использовании интеллектуального редактора эксперт с минимальной помощью инженера по знаниям решает первую и вторую задачи приобретения

знаний, третья и четвертая задачи приобретения знаний выполняются экспертной системой.

Есть надежда, что в перспективе экспертные системы будут приобретать знания аналогично тому, как это делает эксперт-человек (рис. 7.4).

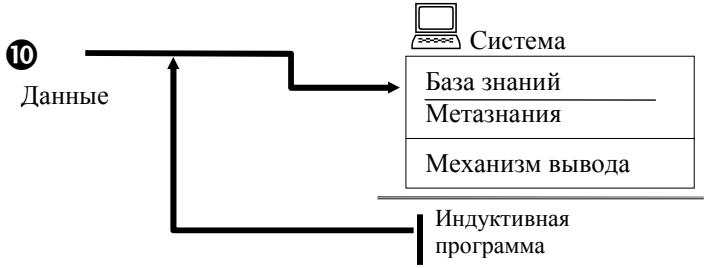


Рис. 7.4

В этом случае индуктивная программа будет анализировать данные, содержащие сведения о некоторой области экспертизы, автоматически формируя значимые отношения и правила, описывающие предметную область. Предполагается, что в базе знаний в явном виде хранятся конкретные факты о предметной области, а задача индуктивной программы – сделать значимые обобщения. Основным достоинством этого подхода является автоматизация всех перечисленных выше четырех задач по приобретению знаний. Экспертных систем, получающих знания исключительно от индуктивных программ, пока нет, имеются лишь экспериментальные программы.

Дальнейшие перспективы развития экспертных систем связываются с приобретением знаний непосредственно из текстов на естественном языке. Модель такого способа приобретения знаний приведена на рис. 7.5.

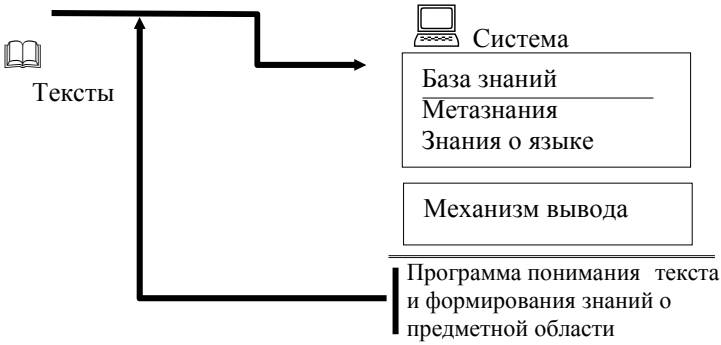


Рис. 7.5

В данном случае требуется читать обычные печатные тексты книги, статьи и извлекать из них знания, т.е. понимать текст, схемы, гра-

фики. Сложность задачи понимания состоит не только в обработке естественного языка, но и в необходимости воссоздать по тексту модель некоторой предметной области. Эти требования пока превосходят возможности существующих программ понимания.

Контрольные вопросы

1. Назовите те характеристики экспертных систем, которые отличают их от обычных программ.
2. Возможно ли применение статистических методов в экспертных системах? Почему?
3. Какие задачи решаются на каждом из этапов разработки экспертных систем?
4. Перечислите и охарактеризуйте стадии существования экспертных систем.

8 ЯЗЫК ПРОГРАММИРОВАНИЯ ПРОЛОГ

8.1 Простейшие программы

Парадигма логического программирования ориентирована на сведение существенной части процесса программирования к описанию объектов, составляющих предметную область, и заданию отношений между этими объектами. Особенностью языка ПРОЛОГ является предоставляемая им возможность определения отношений с акцентом на описание, а не на способ вычисления.

Программирование в логических языках состоит в описании модели рассматриваемой предметной области. Модель строится в терминах объектов и отношений. Отношения между объектами могут быть заданы в виде фактов и правил. *Факт* – это некоторое утверждение, истинность которого считается доказанной.

Простейшая ПРОЛОГ-программа состоит из одного факта и имеет следующий вид:

A.

что интерпретируется следующим образом:

«утверждение ***A*** выполнимо», «факт ***A*** имеет место», «***A*** истинно».

Рассмотрим пример. Пусть предметная область может быть описана двумя фактами:

R – сегодня идет дождь;

S – сегодня идет снег.

Программа ***R.*** означает, что в ПРОЛОГ-систему помещено утверждение «сегодня идет дождь». Факт ***R*** в программе считается доказанным по определению.

Обращение к ПРОЛОГ-программе организовано с помощью *вопросов*. Пример обращения: **? - *R***.

Интерпретация этого предложения такова:

«выполнимо ли утверждение ***R*** ?», «имеет ли место факт ***R*** ?», «истинно ли ***R*** ?»

В ПРОЛОГе принята следующая концепция: «отсутствие факта означает его неудачное выполнение», поэтому, если программа состоит из одного факта «***A.***», а вопрос задан «***?- B***.», то ответом будет «нет» («неудача», «ложь», «No»), если же задан вопрос «***?- A***.», то ответом будет «да» («удача», «истина», «Yes»).

Синтаксически вопрос от факта отличается наличием вначале предложения конструкции «***?-*** ». Вопрос и факт обязательно заканчиваются точкой « . ». Утверждение без точки будем называть *целью*.

Следующий уровень сложности ПРОЛОГ-программы связан с рассмотрением новой конструкции языка, называемой *правилом.* Правило имеет следующий вид: **A:- B_1, B_2, ... , B_n.**

Состоит правило из двух частей, разделенных конструкцией «**:-** », которую можно читать как «если». Левая часть правила («***A***») называется заголовком (головой предложения) и является атомарной формулой. Правая часть правила называется телом правила и представляет собой конечное множество целей, возможно пустое, здесь $B_1, B_2, \dots, B_n$ – факты либо правила, соединенные в единую конструкцию символами « , ». Таким образом, символ « , » означает в теле правила связку «и» (конъюнкцию). Интерпретируется правило следующим образом:

«утверждение ***A*** выполнимо, если выполнимы все $B_1, B_2, \dots, B_n$», «заключение ***A*** имеет место, если имеют место одновременно и B_1, *и* B_2, ..., *и* B_n», «***A*** истинно, если истинны все $B_1, B_2, \dots, B_n$».

Внимание. Правила, также как и факты, и вопросы, обязательно заканчиваются точкой « . ».

Факт можно рассматривать как частный случай правила, когда тело правила пусто ($n = 0$).

Используя понятие *процедуры*, ПРОЛОГ-программу, имеющую вид:

A:- B_1, B_2.
A:- B_3, B_4.

можно прочитать следующим образом: если при выполнении процедуры ***A*** успешно выполнены процедуры B_1 и B_2 , то процедура ***A*** выполнена успешно; если хотя бы одна из процедур B_1 или B_2 завершилась неуспешно, то выполняется второе предложение процедуры ***A***, и соответственно выполняются процедуры B_3 и B_4; если обе эти процедуры закончены успешно, то процедура ***A*** успешно завершена, в противном случае выполнение процедуры ***A*** завершается неуспехом. Таким образом, правила в процедуре связаны связкой «или» (дизъюнкция).

Рассмотрим пример. Пусть программа имеет следующий вид:

A:- B, C.
B:- E.
E.
C.

Задан вопрос: **?- A.**

Ответом на него будет «да», потому что ***B*** выполнимо, а ***C*** – факт, заданный в программе; ***B*** выполнимо, потому что выполнимо ***E***, заданное в программе в виде факта.

ПРОЛОГ-программа и поиск ответа на вопрос могут быть представлены на дереве *И-ИЛИ*. Вершина типа *И* успешно разрешима, если

успешно разрешимы все ее вершины-потомки. Вершина типа *ИЛИ* успешно разрешима, если успешно разрешима хотя бы одна из ее вершин-потомков. Принято отмечать узел *И* дугой, охватывающей все относящиеся к нему ветви, узел *ИЛИ* такой пометки не имеет.

Правило вида A: B_1, B_2, ... , B_n на дереве представляется вершиной типа *И* (рис. 8.1).

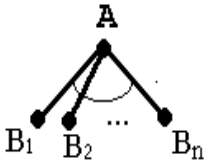


Рис. 8.1

Если в программе встречаются несколько правил с одинаковым заголовком (процедура), то на дереве они представляются вершиной типа *ИЛИ*. Например, правила следующего вида:

A:- B_1, B_2, ... , B_n.
A:- C.
A:- D_1, D_2, ... , D_k.

на дереве будут представлены деревом, изображенным на рис. 8.2.

Заключение ***A*** в общем случае здесь можно получить тремя способами: если удовлетворены все исходные факты или ***B_1, B_2, ... , B_n,*** или ***C,*** или ***D_1, D_2, ... , D_k.*** Расположение узлов в этом дереве отражает то, каким образом можно прийти к заключению ***A***, исходя из различных комбинаций исходных фактов.

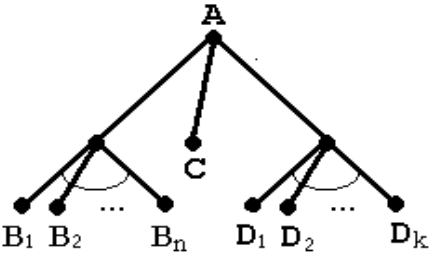


Рис. 8.2

Дерево поиска ответа на вопрос G относительно программы P определяется следующим образом. Корнем дерева является G. Вершины дерева образуют множества целей. Каждая ветвь дерева соответствует ответу на вопрос G относительно программы P. Листья дерева называются успешными вершинами, если множество целей, соответствующее листу, пусто; и безуспешными вершинами, если цель, выделенная в вершине, не может быть выполнена. Успешные вершины соответствуют решениям корня дерева.

В ПРОЛОГе принята стратегия поиска в глубину, когда ведется последовательный обход *И-ИЛИ* дерева сверху вниз и слева направо. Если при обходе дерева потомок вершины типа *ИЛИ* успешно выполнен, то обход поддерева, находящегося справа, приостанавливается и поиск продолжается в глубину, считая успешно выполненной данную *ИЛИ*-вершину. Если потомок вершины типа *И* не выполнен, то обход остальных потомков прекращается, считая безуспешно выполненной

данную *И*-вершину. Поиск продолжается в этом случае со следующей заранее приостановленной *ИЛИ*-вершины. Процесс повторного обращения к вершине, когда обход продолжается с приостановленной *ИЛИ*-вершины, называется *бектрекингом*.

Используя графы *И-ИЛИ*, программу из предыдущего примера и поиск решения можно представить схемой, изображенной на рис. 8.3. Здесь толстыми линиями представлена непосредственно сама программа, а тонкие линии со стрелками показывают последовательность выполнения правил в процессе поиска решения.

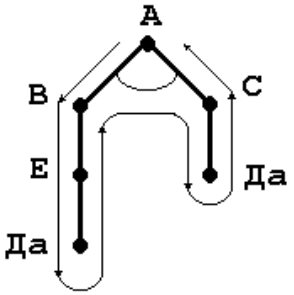


Рис. 8.3

Рассмотрим еще один пример. Пусть имеются утверждения: «Овладеть языком программирования можно (PL), если работать с ним практически (W) и выучить его основы (F). Работа с языком практически возможна, если есть персональный компьютер (P). Выучить основы языка можно при условии работы в библиотеке (L). Выполнены условия работы в библиотеке и наличия персонального компьютера. Следовательно, можно овладеть языком программирования». ПРОЛОГ-программа, задающая описания приведенных выше утверждений, имеет следующий вид:

```
PL:- W, F.
W:- P.
F:- L.
L.
P.
```

Обращение к программе: **?- PL.,** даст ответ «да».

Допустим теперь, что условие работы в библиотеке выполнено, а персонального компьютера нет в наличии. ПРОЛОГ-программа в этом случае будет иметь следующий вид:

```
PL:- W, F.
W:- P.
F:- L.
L.
P:- fail.
```

Здесь автором пособия использован встроенный системный предикат ***fail***, который всегда приводит к безуспешному выполнению правило, в котором ***fail*** использован. Более привычная и наглядная запись отрицания в виде ***not(P)*** невозможна в силу языковых ограничений. Вопрос к программе: **?- PL.** даст ответ «нет».

8.2 Термы

8.2.1 Переменные и константы

Исходными элементами описания предметной области в приведенных выше примерах являются отдельные утверждения: простые – факты и сложные – правила. При этом не рассматривались состав и структура утверждения в смысле связи действия, агента, объекта и других грамматических составляющих предложения. Однако не всегда возможно адекватно описать предметную область таким простым способом. Для углубленного анализа и описания предметной области язык программирования должен иметь возможность работы с константами, переменными и сложными структурами. ПРОЛОГ обладает такими возможностями.

Рассмотрим следующий факт:

«Ходашинский читает курс “Искусственный интеллект”».

Сказуемым или предикатом в данном предложении является «читает курс», этот предикат связывает два объекта «Ходашинский» и «Искусственный интеллект». В языках логического программирования для записи подобных фактов существует стандартная форма записи:

предикат(объект_1, объект_2).

Тогда наш факт может быть записан следующим образом:

lecture(«Ходашинский», «Искусственный интеллект»).

Объекты «Ходашинский» и «Искусственный интеллект» являются здесь константами, отношение с именем ***lecture*** – это предикат. *Резольвента* – это множество целей, которые необходимо выполнить, чтобы получить ответ на поставленный вопрос.

Латинская нотация для предикатов выбрана автором не из большой любви к английскому языку и латинице, а по причине более прозаической: большинство ПРОЛОГ-трансляторов работают только с латинским алфавитом в качестве имен предикатов.

Совокупность фактов называется *базой данных*. Этот термин будет использоваться для объединения фактов при описании предметной области для решения конкретной задачи. Продолжив перечисление читаемых курсов, получим некоторую базу следующего вида:

lecture(«Ходашинский», «Искусственный интеллект»).
l**ecture**(«Адуева», «Программирование»).
lecture(«Хон», «Банки данных»).
lecture(«Лепихина», «АИС»).
lecture («Иванова», «Физика»).
lecture («Байбакова», «Высшая математика»).

Пусть имеется вопрос: «Читает ли Хон курс “Физика”? »
В терминах выбранной нотации ПРОЛОГа этот вопрос выглядит так:

?- **lecture**(«Хон», «Физика»).

Ответ – «нет». Получила ПРОЛОГ-система его следующим образом:

1) из вопроса формируется резольвента, в нашем случае – ***lecture(«Хон», «Физика»)***;

2) выбирается первая цель из резольвенты (она у нас единственная);

3) просматривается база данных в попытке согласовать выбранную цель и факт из базы, в нашем случае такая попытка оказывается безуспешной.

Таким образом, для того чтобы получить положительный ответ на поставленный вопрос, база данных должна содержать факт, удовлетворяющий следующим условиям:

- имя факта и вопроса должны совпадать;
- должна совпадать арность вопроса и факта;
- вопрос и факт должны иметь одни и те же константы на тех же местах.

При выполнении этих условий вопрос удовлетворяет факту и, следовательно, ответ положителен. В нашем случае не нашлось факта, удовлетворяющего вопросу.

Теперь, положим, необходимо знать, а какой же курс читает Хон. Перебор всех вопросов с указанием всех читаемых курсов до получения ответа «да» – это не лучший вариант решения этой проблемы. Эту задачу можно решить, сформулировав вопрос следующим образом:

?- **lecture(«Хон», X).**

Литера ***X*** означает в вопросе неизвестный заранее объект. Такого рода объекты называются переменными. *Переменная* в ПРОЛОГе – это последовательность букв и цифр, начинающаяся с прописной буквы или символа подчеркивания и содержащая только символы букв, цифр и подчеркивания. Переменная, состоящая из одного символа подчеркивания, – это *анонимная переменная*, использующаяся в предложении только один раз. В процессе поиска ответа на вопрос ПРОЛОГ-система просматривает базу данных и пытается найти эквивалентную замену переменной ***X*** объектом из базы, используя при этом алгоритм унификации. Ответом на вопрос: ?- **lecture(«Хон», X)** будет X = «Банки данных».

Переформулируем предыдущую задачу следующим образом: определить, кто читает курс «Банки данных». На вопрос

?- **lecture (X, «Банки данных»).**

система ответит ***X = «Хон»***. Таким образом, отношение ***lecture*** ПРОЛОГ-система может использовать различными способами: если

известны оба его аргумента, то проверяется выполнимость данного отношения на этих аргументах; если неизвестен один из аргументов, неважно какой из них, то система пытается найти эквивалентную замену переменной объектом из базы данных.

Во многих версиях ПРОЛОГа приняты следующие соглашения: заглавные буквы обозначают переменные, а строчные буквы, целые или действительные числа, или любые символы, заключенные в кавычки, обозначают постоянные значения – *константы*. Этих же соглашений будем придерживаться и мы в дальнейшем изложении.

Добавим в базу данных факты, указывающие место работы преподавателей:

worker(«Ходашинский», «АОИ»).
worker(«Адуева», «АОИ»).
worker(«Хон», «АОИ»).
worker(«Лепихина», «АОИ»).
worker(«Иванова», «Физика»).
worker(«Байбакова», «Высшая математика»).

Предположим, что необходимо ответить на вопрос: «На какой кафедре читается курс “Программирование”?» Один из путей решения этой задачи – сначала найти преподавателя, читающего курс «Программирование», а затем определить место работы этого преподавателя. В ПРОЛОГе это можно сделать в одном вопросе, воспользовавшись *общей переменной* и соединив оба вопроса, используя операцию конъюнкции: ?- **lecture**(X, «Программирование»), **worker**(X, Y). Этот вопрос читается следующим образом: «Существуют ли такие ***X*** и ***Y***, что **lectu-re**(X, «Программирование») и **worker**(X, Y) выполнимы одновременно?»

Рассмотрим действия ПРОЛОГ-системы при выполнении данного вопроса:

1) формируется резольвента из двух целей: **lecture**(X, «Программирование»), **worker**(X, Y);

2) выбирается первая цель из резольвенты – **lecture**(X, «Программирование»);

3) просматривается база данных в попытке согласовать выбранную цель и факт из базы, попытка оказывается успешной, в базе найден факт **lecture**(«Адуева», «Программирование»);

4) переменная ***X*** с этого момента конкретизирована и ей соответствует значение «Адуева»;

5) первая цель резольвенты выполнена, система переходит к следующей цели – **worker**(Адуева», Y);

6) просматривается база данных в попытке согласовать выбранную цель и факт из базы, попытка оказывается успешной, в базе найден факт **worker**(«Адуева», «АОИ»);

7) переменной ***Y*** присваивается значение «АОИ»;

8) в резольвенте отсутствуют невыполненные цели, значит, вопрос успешно выполнен, система выдает ответ: X = «Адуева», Y = «АОИ».

Фундаментальными операциями над переменными является конкретизация и унификация, с помощью которых выполняются однократное присваивание и передача параметров.

Сформулируем предыдущую задачу в общем виде:

«Определить кафедру, на которой читается заданный курс».

Запрос к базе данных оформим в виде следующего правила:

quest_1(X, Y):- **lecture**(Z, X), **worker**(Z, Y).

Правило определяет предикат ***quest_1,*** имеющий два аргумента. Предикат ***quest_1(X, Y)*** истинен, если установлено взаимно-однозначное соответствие между читаемым курсом и кафедрой.

Предыдущий вопрос будет записан следующим образом:

?- **quest_1**(«Программирование», Y).

Действия ПРОЛОГ-системы здесь следующие:

1) формируется резольвента **quest_1**(«Программирование», Y);

2) выбирается цель из резольвенты, т.е. **quest_1** («Программирование», Y);

3) делается попытка согласовать выбранную цель и заголовок правила из базы, попытка оказывается успешной;

4) переменная ***X*** с этого момента конкретизирована, и ей соответствует значение «Программирование», переменная ***Y*** не конкретизирована;

5) формируется новая резольвента **lecture**(Z, «Программирование»), **worker**(Z, Y);

6) дальнейшие действия совпадают с действиями ПРОЛОГ-системы при решении предыдущей задачи с той лишь разницей, что вместо переменной X здесь используется Z.

На вопрос: ?- **quest_1**(«Сети», Y) будет дан ответ «нет», потому что попытка согласовать выбранную из резольвенты цель **lecture**(Z, «Сети») и факт из базы данных оказывается безуспешной.

Рассмотрим вхождения переменных в правило. Всякий раз, как переменой присваивается значение, все вхождения переменной в правило становятся конкретизированными, т.е. любая попытка присвоить конкретизированной переменой другое значение закончится неудачей, изменить это значение можно только после очередного обращения к правилу. В связи с этим, в логическом программировании теряет смысл весьма распространенная конструкция императивного програм-

мирования вида ***N = N+1***, такая конструкция всегда приводит к отказу. Таким образом, областью действия переменной в правиле является это правило. Исключением является анонимная переменная. Она используется в случае, когда переменная встречается в утверждении только один раз и ее значение не играет никакой роли в данном утверждении. Синтаксически анонимная переменная задается одним символом подчеркивания «_». Например, используя приведенную выше базу данных, необходимо определить, есть ли преподаватель по фамилии «Петров». Для этого можно вопрос сформулировать следующим образом: ?- **worker**(«Петров», _). Семантика вопроса такова: определить, есть ли в базе данных объект «Петров», не определяя кафедры, на которой он работает.

8.2.2 Сложные термы

Дополним базу данных фактами о месте и времени прочтения лекции, а также имени группы, в которой эта лекция читается:

room_time_group(«Искусственный интеллект», «426_f», 9, 11, «вторник», «4321»).
room_time_group(«Искусственный интеллект», «426_f», 9, 11, «вторник», «4322»).
room_time_group(«Программирование», «426_f», 9, 11, «среда», «4321»).
room_time_group(«Программирование», «426_f», 9, 11, «среда», «4322»).
room_time_group(«Банки данных», «426_f», 11, 13, «среда», «4321»).
room_time_group(«Банки данных», «426_f», 11, 13, «среда», «4322»).
room_time_group(«АИС», «426_f», 14, 16, «четверг», «4321»).
room_time_group(«АИС», «426_f», 14, 16, «четверг», «4322»).
room_time_group(«Физика», «421_f», 11, 13, «вторник», «4341»).
room_time_group(«Физика», «421_f», 11, 13, «вторник», «4342»).
room_time_group(«Высшая математика», «421_f», 14, 16, «вторник», «4341»).
room_time_group(«Высшая математика», «421_f», 14, 16, «вторник», «4342»).

Время прочтения лекции здесь состоит из трех компонент: времени начала лекции, времени окончания лекции и дня недели. Эти компоненты могут быть объединены в одну структуру, для этого необходимо выбрать имя структуры и определить входящие в нее компоненты. Для нашего случая можно создать следующую структуру:

time(«время начала лекции», «время окончания лекции», «день недели»),

здесь «время начала лекции» – это целое число;

«время окончания лекции» – также целое число;

«день недели» – последовательность букв, заключенная в кавычки.

Например, **time(**9, 11, «вторник»). Структурированные таким образом объекты называются *сложным термом*. Имя сложного терма будем называть *функтором*, компоненты – *аргументами*, число аргументов в терме – *арностью*.

Приведенный выше фрагмент базы данных будет выглядеть следующим образом:

room_time_group(«Искусственный интеллект», «426_f», **time(**9, 11, «вторник»), «4321»).
room_time_group(«Искусственный интеллект», «426_f», **time(**9, 11, «вторник»), «4322»).
room_time_group(«Программирование», «426_f», **time(**9, 11, «среда»), «4321»).
room_time_group(«Программирование», «426_f», **time(**9, 11, «среда»), «4322»).
room_time_group(«Банки данных», «426_f», **time(**11, 13, «среда»), «4321»).
room_time_group(«Банки данных», «426_f», **time(**11, 13, «среда»), «4322»).
room_time_group(«АИС», «426_f», **time(**14, 16, «четверг»), «4321»).
room_time_group(«АИС», «426_f», **time(**14, 16, «четверг»), «4322»).
room_time_group(«Физика», «421_f», **time(**11, 13, «вторник»), «4341»).
room_time_group(«Физика», «421_f», **time(**11, 13, «вторник»), «4342»).
room_time_group(«Высшая математика», «421_f», **time(**14, 16, «вторник»), «4341»).
room_time_group(«Высшая математика», «421_f», **time(**14, 16, «вторник»), «4342»).

Сложный терм в программе ведет себя как единый объект данных. Так например, на вопрос

?- **room_time_group**(«Искусственный интеллект», _, X, _).

система выдаст ответ: X = **time**(9, 11, «вторник»).

Аргументы в сложном терме могут быть не только константами, но и переменными, например, вопрос «По каким дням недели читается курс “Искусственный интеллект”» будет выглядеть следующим образом:

?- **room_time_group**(«Искусственный интеллект», _ , **time**(_, _, X), _).

Ответ: X = «вторник».

Константы, переменные, сложные термы в ПРОЛОГе так же, как и в логике предикатов первого порядка, называются *термами*. Так же, как и в логике, фундаментальной операцией над термами является операция унификации. Два терма унифицируемы в следующих случаях:

- термы абсолютно одинаковы;
- первый терм – переменная, второй – любой терм;
- когда у них одни и те же функтор и арность и все их аргументы унифицируемы.

8.3 Поиск решения

Неформально поиск решения в логической программе может быть описан следующим образом. Он начинается с некоторого исходного вопроса и завершается одним из двух результатов: успешным завершением или отказом. Для данного вопроса может существовать несколько успешных решений, дающих различные результаты. Кроме того, может иметь место бесконечный поиск (зацикливание), с которым не связываются никакие результаты. Поиск решения проводится с помощью редукции цели. На каждом этапе имеется некоторая резольвента. Выбирается такая цель в резольвенте и такое предложение в логической программе, что заголовок предложения унифицируем с целью. Поиск решения продолжается с новой резольвентой, полученной из старой заменой выбранной цели на тело выбранного предложения и последующим применением наиболее общего унификатора заголовка предложения и выбранной цели. Поиск решения завершается, если резольвента пуста. В этом случае говорят, что решение найдено программой.

Рассмотрим следующие утверждения:

Все люди смертны.

Сократ – человек.

Следовательно, Сократ – смертен.

Обозначим ***man(X)*** – ***X*** является человеком, ***mortal(X)*** – ***X*** смертен. ПРОЛОГ-программа, выражающая приведенные выше утверждения, будет иметь следующий вид:

```
mortal(X):-man(X).
man(«Сократ»).
?-mortal(«Сократ»).
```

Поиск решения начинается с резольвенты ***mortal(«Сократ»).*** В программе ищется правило с именем ***mortal*** и производится унификация переменой ***X*** и константы ***«Сократ».*** Новая резольвента, полученная из старой заменой выбранной цели на тело выбранного предложения и применением унификатора ***X*** = ***«Сократ»,*** будет ***man(«Сократ»).*** В программе ищется правило с именем ***man.*** Унификация резольвенты и выбранного предложения проходит успешно, а так как у выбранного предложения нет тела, то новая резольвента пуста, и вычисление успешно заканчивается.

Рассмотрим следующую программу:

```
p(X):-q(X), s(X), r(X).
p(X):-u(X).
q(a).
q(b).
s(c).
```

s(b).
s(d).
r(a).
r(c).
u(d).
u(a).
?- p(a).

Начальная резольвента – ***p(a).*** Таким образом, имеется единственная резольвента, она и выбирается для редукции. Выбирается первое предложение в логической программе, и оно унифицируется с резольвентой: **p(X):-q(X),s(X), r(X).** Унификатор цели и заголовка имеет следующий вид: **X=a.** Новая резольвента **q(X), s(X), r(X)** после применения унификатора будет иметь следующий вид: **q(a), s(a), r(a).**

На следующей итерации из резольвенты выбирается первая цель, а именно ***q(a).*** В программе выбирается предложение для отношения ***q***, первым из таковых является ***q(a).*** Так как у выбранного предложения нет тела, то цель ***q(a)*** выполнена. Из резольвенты выбирается следующая цель – ***s(a).*** В программе выбирается предложение для отношения ***s***. Однако выполнить цель ***s(a)*** не удается. Выполнение резольвенты ***q(a), s(a), r(a)*** заканчивается неудачей. После бектрекинга выбирается второе предложение в логической программе для унификации с начальной резольвентой ***p(a)***: **p(X):-u(X).**

Новая резольвента **u(X)** после применения унификатора ***X=a*** будет иметь следующий вид: **u(a).** Из резольвенты выбирается цель ***u(a).*** В программе выбирается предложение для отношения ***u***, а именно ***u(a).*** Так как у выбранного предложения нет тела, то новая резольвента пуста, и вычисление успешно заканчивается, ПРОЛОГ-система выдает ответ «да».

Представим данную программу и поиск решения в виде дерева (рис. 8.4).

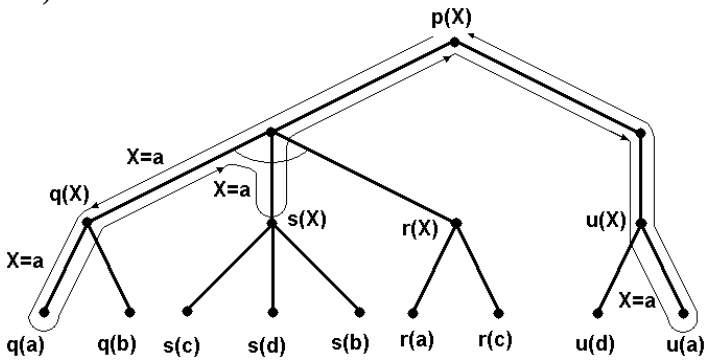


Рис. 8.4

Изменим вопрос к программе из предыдущего примера, пусть вопрос будет иметь вид: **?- p(Y).** Начальная резольвента – ***p(Y),*** она и выбирается для редукции. Ищется первое предложение в логической программе и унифицируется с резольвентой: **p(X):-q(X),s(X), r(X).** Унификатор цели и заголовка имеет следующий вид: **Y=X.** Новая резольвента будет иметь следующий вид: **q(X), s(X), r(X).**

На следующей итерации из резольвенты выбирается первая цель, а именно ***q(X).*** В программе выбирается предложение для данного отношения – ***q(a).*** После унификации, когда ***X*** конкретизируется и получает значение ***a***, цель ***q(a)*** в резольвенте выполнена. Из резольвенты выбирается следующая цель – ***s(X),*** которая после применения унификатора примет вид – ***s(a).*** В программе ищется предложение для отношения ***s***. Однако выполнить цель ***s(a)*** не удается. После бектрекинга выбирается второе предложение в логической программе для отношения ***q*** – ***q(b).*** После унификации ***X*** конкретизируется и получает значение ***b***, цель ***q(b)*** в резольвенте выполнена. Из резольвенты выбирается следующая цель – ***s(X),*** которая после применения унификатора примет вид – ***s(b).*** В программе ищется предложение для отношения ***s*** – ***s(b),*** цель ***s(b)*** в резольвенте выполнена. Из резольвенты выбирается последняя цель – ***r(X),*** которая после применения унификатора примет вид – ***r(b),*** но выполнение ее заканчивается неудачей. Таким образом, первое предложение логической программы **p(X):-q(X),s(X), r(X)** выполнить не удалось. После бектрекинга система возвращается к начальной резольвенте ***p(Y)*** и пытается унифицировать ее со вторым предложением программы **p(X):-u(X).**

Новая резольвента на данном шаге будет: **u(X),** а унификатор – **Y=X**. Из резольвенты выбирается цель ***u(X).*** В программе выбирается предложение для отношения ***u***, а именно ***u(d),*** после унификации, когда ***X*** конкретизируется и принимает значение ***d*** , цель выполнена, ***Y*** принимает значение ***d***. ПРОЛОГ-система пытается найти все возможные решения, поэтому она возвращается вновь к цели ***u(X)*** и ищет в программе другие отношения ***u***, а именно ***u(a),*** после унификации, когда ***X*** принимает значение ***a***, цель выполнена, ***Y*** принимает значение ***a***. Таким образом, ответом на вопрос: **?- p(Y).** будут два решения **Y=d, Y=a.**

Таким образом, поиск решения в ПРОЛОГ-программе выполняется от исходного вопроса к фактам, т.е. в обратном направлении. Если предложений, подлежащих унификации, несколько, то производится унификация с первым из них. Если данное предложение выполнено, то программа выполняется дальше. Если же данное предложение не выполнено, то после бектрекинга выполняются другие предложения, подлежащие унификации.

Контрольные вопросы

1. Чем принципиально отличаются языки логического программирования от традиционных языков программирования?
2. Какую роль играет переменная в логическом языке программирования? Какова область ее действия?
3. Как задается сложный терм?
4. Что такое цель в логическом программировании?
5. Как задается и выполняется ПРОЛОГ-программа? Перечислите типы ответов на вопросы.

9 ТЕХНИКА ПРОЛОГ-ПРОГРАММИРОВАНИЯ

9.1 Рекурсия и итерация

Один из стандартных приемов решения сложной задачи состоит в том, чтобы разбить задачу на более простые подзадачи, решить эти простые задачи, а затем объединить эти подзадачи для получения общего решения. Когда подход к решению подзадачи совпадает с подходом к решению задачи в целом, тогда такой процесс называется *рекурсией*. Она используется как метод описания программ, выполнению которых предшествует выполнение их собственных копий. Рекурсия является одним из основных приемов программирования на ПРОЛОГе.

В предыдущем разделе рассматривались правила, описывающие новые отношения в терминах существующих отношений. Важным расширением множества подобных правил являются рекурсивные определения, в которых отношения определяются в терминах этих же отношений. Таким образом, процедуру прямо или косвенно обращающуюся к себе, называют *рекурсивной*. Рекурсивное определение должно обязательно содержать утверждение, прерывающее процесс рекурсивного обращения. Такое утверждение называют *граничным условием*.

Рассмотрим программу проверки связности в ориентированном графе (орграфе). Орграф можно задать с помощью набора фактов следующего вида: **edge(a, b).**

Указанный факт входит в программу, если в графе существует ребро, ведущее из вершины ***a*** в вершину ***b***. Пусть дан граф (рис. 9.1).

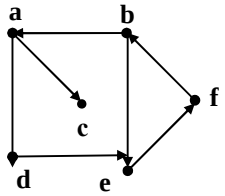


Рис. 9.1

Фрагмент логической программы, описывающей указанный граф, имеет вид:

edge (a, d). edge (b, a). edge (d, e).
edge (e, f).
edge (a, c). edge (b, e). edge (f, b).

Две вершины в орграфе связаны, если существует последовательность ребер, ведущая из первой вершины во вторую. Таким образом, цель ***connected(B1, B2)*** выполнима, если ***B1*** и ***B2*** связаны. Тогда процедура, описывающая связность, будет иметь следующий вид:

connected(B1, B2):- edge(B1, B2).
connected(B1, B2):- edge(B1, X), connected(X, B2).

Первое предложение в процедуре – это *граничное условие*, смысл этого утверждения заключается в следующем: вершины ***B1*** и ***B2*** свя-

заны, если в графе есть ребро, определенное фактом ***edge(B1, B2).*** Второе предложение в процедуре – это *рекурсивное условие*, смысл этого утверждения заключается в следующем: вершины ***B1*** и ***B2*** связаны, если в графе существует последовательность ребер ***edge(B1, X_1), edge(X_1, X_2), ..., edge(X_{N-1}, X_N), edge(X_N, B2),*** ведущая из ***B1*** в ***B2***.

Ответом на вопрос: **?- connected(a, f).** будет «да».

Рассмотрим процесс получения данного ответа. Вначале ПРОЛОГ-система пытается унифицировать запрос ***connected(a, f)*** с первым предложением процедуры. Так как нет ребра, ведущего из вершины ***a*** в вершину ***f*** (в программе отсутствует факт ***edge (a, f)***), то попытка заканчивается неудачей. Далее ПРОЛОГ-система пытается унифицировать запрос **connected(a, f)** со вторым предложением процедуры. После унификации запроса с головой второго правила переменные ***B1*** и ***B2*** получат значение ***a*** и ***f*** соответственно. Далее ПРОЛОГ-система переходит к выполнению целей из тела второго правила:

edge(a, X), connected(X, f).

Цель ***edge(a, X)*** унифицируется с фактом ***edge (a, d),*** а переменной ***X*** присваивается значение ***d***. Выполнение цели ***connected(d, f)*** приводит ко второму рекурсивному обращению.

При втором рекурсивном обращении ПРОЛОГ-система пытается унифицировать запрос ***connected(d, f)*** с первым предложением процедуры. Эта попытка неудачна, поскольку нет ребра, ведущего из вершины ***d*** в вершину ***f*** (в программе отсутствует факт ***edge (d, f)).*** Попытка унифицировать цель ***connected(d, f)*** со вторым предложением процедуры удачна. Для того чтобы отличить второе обращение к предложению от первого, переменным в предложении присвоим индекс ***2***. Примем следующее соглашение: индекс ***n*** у переменной показывает, что она используется при ***n***-ом рекурсивном обращении. После унификации цели ***connected(d, f)*** с головой второго правила переменные ***$B1_2$*** и ***$B2_2$*** получат значения ***d*** и ***f*** соответственно и ПРОЛОГ-система переходит к выполнению целей из тела второго правила:

edge(d, X_2), connected(X_2, f).

Цель ***edge(d, X_2)*** унифицируется с фактом ***edge (d, e)***, а переменной ***X_2*** присваивается значение ***e***. Выполнение цели **connected(e, f)** приводит к третьему рекурсивному обращению.

Третье рекурсивное обращение приводит к унификации цели ***connected(e, f)*** с первым предложением процедуры. Переменные ***$B1_3$*** и ***$B2_3$*** в этом предложении получат значения ***e*** и ***f*** соответственно и ПРОЛОГ-система переходит к поиску в программе факта ***edge (e, f).*** Такой факт присутствует в программе, и выполнение первого предложения процедуры закончится успешно. Выполнено граничное условие,

рекурсивных обращений нет, задача определения связности решена, ПРОЛОГ-система выдаст ответ «да».

На рисунке 9.2 приведена трассировка запроса.

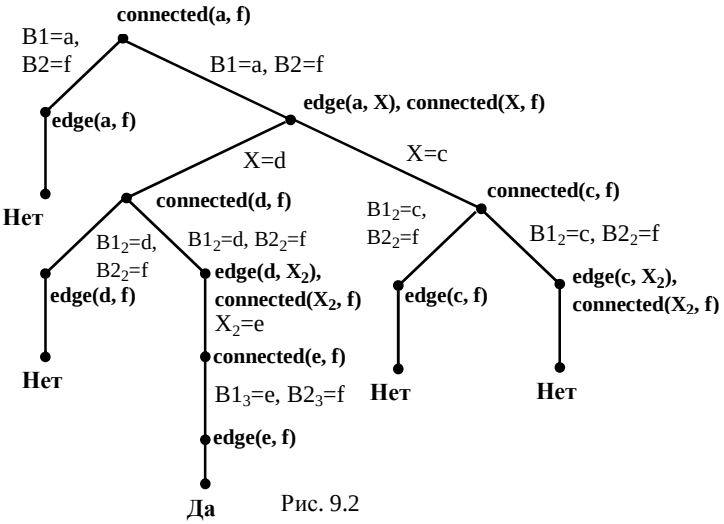


Рис. 9.2

Здесь вершины – это отношения, участвующие в вычислении, а ребра помечены значениями переменных после унификации. Так как ПРОЛОГ-система ищет все возможные решения, заданные программой, то на рисунке приведен также и безуспешный поиск решения после присвоения переменной ***X*** значения ***c***.

Как видно из рисунка, указанный запрос имеет единственное решение (ответ «да»). Однако я не рекомендуется пользоваться процедурой ***connected(B1, B2)*** в практических приложениях, потому как при проверке связности в графе с циклами эта программа работу не заканчивает. Например, на вопрос: **?- connected(a, h).** ответ получен не будет. В нашем случае дерево поиска цели относительно приведенной выше программы содержит бесконечную ветвь, вычисление не завершается. Протокол вычисления приведен ниже:

connected(a, h)
edge (a, d)
connected(d, h)
edge (d, e)
connected(e, h)
edge (e, f)
connected(f, h)
edge (f, b)

connected(b, h)
edge (b, a)
connected(a, h)
edge (a, d)
. . . . . .

Процедура ***connected*** приведена здесь исключительно для пояснения процесса конструирования рекурсивного определения. В разделе, посвященном графам, приведем процедуру проверки связности, пригодную для практического применения.

В общем случае ПРОЛОГ-система находит ответ на поставленный запрос в несколько этапов. На первом этапе (*этап разбиения*), используя принцип редукции поставленной задачи, система отвечает на вопрос, возможно ли вообще решение задачи, откладывая вычисление некоторых утверждений на более поздние этапы. После завершения редукции задачи система возвращается к ранее отложенным утверждениям и находит конкретный ответ на поставленную задачу (*этап вычисления*).

Рассмотрим программу вычисления факториала числа *n*. Операция вычисления факториала *n*! рекурсивно определяется следующим образом:

0! = 1.

n! = *n**(*n*-1)!.

Введем отношение ***factor(N, F),*** где *N* – целое неотрицательное число, *F* – факториал числа *N*. Исходя из определения операции вычисления факториала, *граничное условие* будет иметь следующий вид:

factor(0, 1).

Рекурсивное определение можно сформулировать следующим образом: факториал целого неотрицательного числа n вычисляется как произведение указанного числа n на факториал числа (*n*-1):

factor(N, F):- N > 0, N1 = N-1, factor(N1, F1), F = N * F1.

Ответом на вопрос: **?- factor(3, A).** будет 6.

Рассмотрим процесс получения данного ответа. Вначале ПРОЛОГ-система пытается унифицировать запрос ***factor(3, A)*** с первым предложением процедуры. Так как невозможно конкретизировать константы ***3*** и ***0,*** то попытка заканчивается неудачей. Далее ПРОЛОГ-система пытается унифицировать запрос ***factor(3, A)*** со вторым предложением процедуры. После унификации запроса с головой второго правила переменная ***N*** конкретизируется и получит значение ***3,*** а переменная ***A*** получат значение ***F***. Далее ПРОЛОГ-система переходит к выполнению целей из тела второго правила:

3 > 0, N1 = 3-1, factor(N1, F1), F = 3 * F1.

Выполнение предиката ***3 > 0*** заканчивается успешно. Выполнение предиката ***N1 = 3-1*** приводит присвоению переменной ***N1*** значения ***2***.

Обращение к цели ***factor(2, F1)*** приводит ко второму рекурсивному обращению. Вычисление утверждения ***F = 3 * F1*** откладывается.

При втором рекурсивном обращении ПРОЛОГ-система пытается унифицировать цель ***factor(2, F1)*** с первым предложением процедуры. Эта попытка неудачна, поскольку ***2*** неравно ***0.*** Попытка унифицировать цель ***factor(2, F1)*** со вторым предложением процедуры удачна. Для того чтобы отличить второе обращение к предложению от первого, переменным в предложении присвоим индекс ***2***. После унификации цели ***factor(2, F1)*** с головой второго правила переменные N_2 и ***F1*** конкретизируются, получая значения ***2*** и F_2 соответственно, и ПРО-ЛОГ-система переходит к выполнению целей из тела второго правила:

$$2 > 0,\ N1_2 = 2\text{-}1,\ factor(N1_2, F1_2),\ F1 = 2 * F1_2.$$

Выполнение предиката ***2 > 0*** заканчивается успешно. Выполнение предиката $N1_2 = 2\text{-}1$ приводит присвоению переменной $N1_2$ значения ***1***. Обращение к цели ***factor(1,*** $F1_2$***)*** приводит к третьему рекурсивному обращению. Вычисление утверждения $F1 = 2 * F1_2$ откладывается.

Третье рекурсивное обращение начинается с неудачной попытки унифицировать цель ***factor(1,*** $F1_2$***)*** с первым предложением процедуры. Унификация со вторым предложением процедуры заканчивается удачно. После унификации цели ***factor(1,*** $F1_2$***)*** с головой второго правила переменные N_3 и $F1_2$ получат значение ***1*** и F_3 соответственно, и ПРОЛОГ-система переходит к выполнению целей из тела второго правила:

$$1 > 0,\ N1_3 = 1\text{-}1,\ factor(N1_3, F1_3),\ F_2 = 1 * F1_3.$$

Выполнение предиката ***1 > 0*** заканчивается успешно. Выполнение предиката $N1_3 = 1\text{-}1$ приводит присвоению переменной $N1_3$ значения ***0***. Обращение к цели ***factor(0,*** $F1_3$***)*** приводит к четвертому рекурсивному обращению. Вычисление утверждения $F_2 = 1 * F1_3$ откладывается.

Четвертое рекурсивное обращение приводит к унификации цели ***factor(0,*** $F1_3$***)*** с первым предложением процедуры. Переменная $F1_3$ после конкретизации получит значение ***1.*** Выполнено граничное условие, рекурсивных обращений нет, задача вычисления факториала решена; ПРОЛОГ-система выдаст ответ «да» и перейдет к вычислению отложенных утверждений, пытаясь конкретизировать самое последнее из них.

Самое последнее из отложенных утверждений – это $F1_2 = 1 * F1_3$. Так как переменная $F1_3$ равна 1, то в результате конкретизации переменная $F1_2$ получает значение ***1***. Предпоследнее отложенное утверждение – это $F1 = 2 * F1_2$, здесь переменная ***F1*** равна ***2***. Предыдущее из отложенных утверждений – это ***F = 3 * F1.*** Поскольку ***F1 = 2,*** то ***F = 6***. Но на первом шаге переменная ***A*** была унифицирована с переменной ***F***, в результате ***A*** получает значение, равное ***6***.

На рис. 9.3 приведена трассировка запроса.

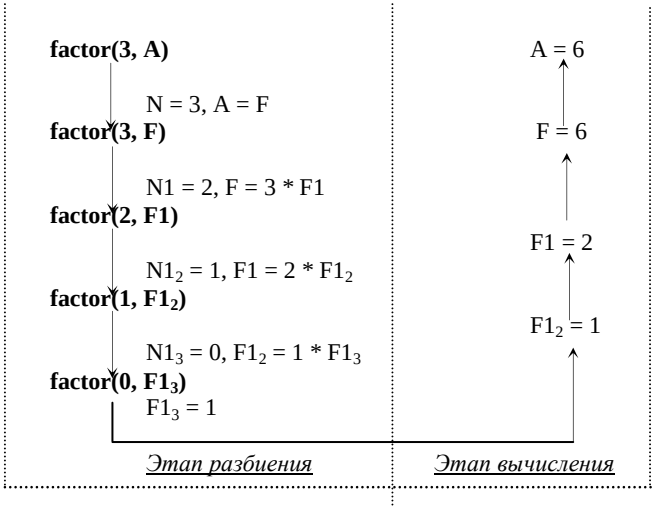


Рис. 9.3

Решение задачи вычисления факториала можно представить в виде протокола:

factor(3, A)	A = 6
factor(3, F)	F = 6
factor(2, F1)	F1 = 2
factor(1, F1$_2$)	F1$_2$ = 1
factor(0, F1$_3$)	F1$_3$ = 1
Да	

С точки зрения реализации ПРОЛОГ-системы, все отложенные утверждения должны храниться в стековой памяти. На этапе разбиения система их помещает туда, а на этапе вычисления обращается к ним и производит определенные операции над ними.

В ПРОЛОГе отсутствуют конструкции типа **for … do, while … do, repeat … until,** задающие циклическое выполнение программ. Вместо них используются рекурсивные конструкции. Рекурсия используется в ПРОЛОГе и для организации итерационных циклов, которые характеризуются последовательным приближением к искомому значению. Главное отличие итерации от рекурсии состоит в том, что при использовании первой отсутствуют отложенные вычисления, а значит, нет необходимости в использовании дополнительной памяти стека. В общем случае объем памяти при рекурсивной реализации линейно зависит от числа выполненных рекурсивных обращений, в то время как объем памяти для выполнения итераций ограничен конкрет-

ным значением и не зависит от числа обращений. Почти все рекурсивные программы, выполняющие арифметические вычисления, могут быть представлены в виде итерационных программ.

Рассмотрим применение итерации на примере вычисления факториала.

factorial(N, F):- N>=0, factor1(0, N, 1, F).
factor1(I, N, X, F):- I<N, K = I+1, Z = X*K, factor1(K,N,Z,F).
factor1(N, N, F, F).

Основной итерационный цикл здесь реализован в процедуре ***factor1***, в которой первый аргумент – это счетчик итераций, третий аргумент используется для накопления текущего значения факториала, второй и четвертый аргументы – это целое и его факториал, соответственно. *Граничное условие*, соответствующее второму предложению процедуры, формулируется следующим образом: как только счетчик итераций достигнет значения целого *N*, значение факториала будет равно значению накопленной переменной. *Рекурсивное условие*: пока счетчик не достиг *N*, вычислять факториал от значения счетчика и запоминать это значение в промежуточной переменной.

Задав к программе вопрос, **factorial(3, A).** получим ответ **A** = 6. На рис. 9.4 приведена трассировка запроса.

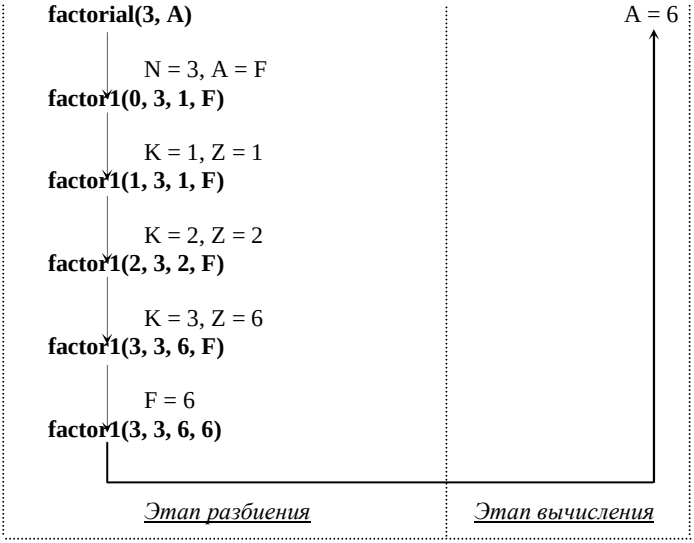


Рис. 9.4

Этап вычисления в итерационных программах, по сути, отсутствует.

Итерационную процедуру вычисления факториала можно задать иным способом, когда счетчик будет меняться не от 0 до *N*, а от *N* до 0.

```
factorial(N, F):- factor1(N, 1, F).
factor1(N, X, F):- N>=0, K = N-1,  Z = X*N, factor1(K, Z, F).
factor1(0, F, F).
```

Здесь процедура содержит меньшее количество аргументов, а число итераций и объем вычислений не изменились по сравнению с предыдущей процедурой, а это позволяет сделать вывод о том, что данная процедура более эффективна, нежели предыдущая.

Таким образом, для организации итерационного цикла необходимо выполнить следующие действия:

- задать начальное значение переменной (одной или нескольких), изменяющихся в цикле;
- изменить переменную перед каждым новым повторением цикла;
- проверить условие окончания или повторения цикла.

Первый пункт выполняется при первоначальном обращении к итерации. Второй пункт реализуется рекурсивным условием, и наконец, последний пункт обеспечивается граничным условием.

9.2 Списки

В ПРОЛОГе имеется единственная структура данных – логический терм. С этой точки зрения, список – это терм арностью два, в котором второй аргумент является также списком. Это очень важная и интересная структура, а потому ей посвящен отдельный раздел.

С математической точки зрения, список – это конечная упорядоченная последовательность элементов. Список в языках программирования – это бинарная структура или специальный вид сложного терма, состоящий из головы списка и хвоста списка, где голова – это элемент списка, а хвост – это список. Элементы списка, в свою очередь, сами могут быть списками. Функтор для обозначения списка принят следующий:

.(X, L),

однако чаще используется инфиксная форма записи следующего вида:

[X | L],

где ***X*** – голова, ***L*** – хвост, операция «|» обозначает слияние элемента-головы и элемента-хвоста списка в единый список. Операцию «|» можно интерпретировать и как операцию разбиения списка на голову и хвост. Таким образом, список в ПРОЛОГе – это такая структура, в которой операции включения /исключения производятся только в на-

чале этой структуры. Для элементов списка справедливо следующее правило: «Первым пришел – последним ушел».

На вопрос **?- L = [a | [b, c, d]].** будет дан ответ **L = [a, b, c, d].**

При построении списков используется константный символ **[]**, обозначающий пустой список. Важное свойство пустого списка – его нельзя разделить на голову и хвост, попытка выполнить такую операцию приведет к неудаче.

Дадим рекурсивное определение списка:

- **[]** – это список;
- пусть ***L*** – список, ***A*** – терм, тогда **[*A* | *L*]** – список.

На вопрос **L = [a , b, c, d | []].** будет дан ответ **L = [a, b, c, d].**

Элементами списка могут быть не только константы, но любые термы – переменные, структуры, включая другие списки. Важным свойством списка является его динамичность, т.е. отсутствие необходимости заранее описывать размер списка. Переменные в списке ничем не отличаются от переменных в любой другой структуре, с ними можно проводить те же действия, что и с переменными вне списка.

На вопрос

?- L = [a , b, c, d], L = [X | Y], Y = [V, Z | W].

будет дан ответ

L = [a, b, c, d], X = a, Y = [b, c, d], V = b, Z = c, W = [d].

А вот на вопрос **?- L = [], L = [X | Y].** будет дан ответ «нет», так как пустой список нельзя разделить на голову и хвост.

Таким образом, список в ПРОЛОГ-программе, если он не пуст, можно представить различными способами:

- перечислением всех элементов списка – **[a, b, c, d], [X, Y, Z, V, W];**
- перечислением нужного количества элементов начала списка и ссылкой на его хвост – **[a , b, c | L]** или **[X , Y | L]**;

Фундаментальной операцией на списках является определение вхождения отдельного элемента в список, дадим имя этому отношению ***member(X, L),*** где ***X*** – это элемент, ***L*** – это список. Сформулируем *граничное условие*: элемент содержится в списке, если он совпадает с головой этого списка. *Рекурсивное условие*: элемент содержится в списке, если он является элементом хвоста этого списка.

Программа, рекурсивно определяющая отношение принадлежности элемента списку, имеет вид:

```
member(X, [X | _]).
member(X, [_ | Y]):- member(X, Y).
```

Ответом на вопрос **?- member(a, [b, a, c]).** будет «да».

Рассмотрим процесс получения данного ответа. Начальная резольвента ***member(a, [b, a, c]).*** В программе выбирается первое предложение и делается попытка унифицировать его с резольвентой. Попытка заканчивается неудачей, так как переменной ***X*** одновременно не может быть присвоено значение ***a*** (первый аргумент), ***b*** (голова списка второго аргумента). Далее в программе выбирается второе предложение и делается попытка унифицировать его с резольвентой. Попытка удачна, новая и единственная резольвента – ***member(a, [a,c]).*** В программе выбирается первое предложение, и оно успешно унифицируется с резольвентой. Первое предложение программы не имеет тела, резольвента пуста, решение найдено. Но резольвента ***member(a, [a,c])*** может быть унифицирована и со вторым предложением программы, после унификации получаем новую резольвенту ***member(a, [c]).*** Данная резольвента не может быть унифицирована с первым предложением программы, а вот со вторым предложением унификация заканчивается успешно, новая резольвента – ***member(a,*** **[]** ***),*** которая не может быть унифицирована ни с первым, ни со вторым предложением. Таким образом, был получен один положительный ответ.

Графическое представление поиска решения приведено на рис. 9.5.

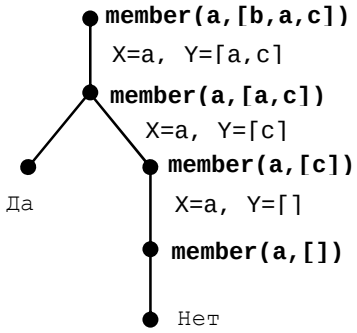


Рис. 9.5

Не менее важной операцией над списками является операция соединения двух списков для получения третьего, дадим имя этому отношению ***append(L1, L2, L3),*** здесь ***L3*** – это конкатенация списков ***L1*** и ***L2***. *Граничное условие*: результатом соединения пустого списка со списком ***L*** будет список ***L***. *Рекурсивное условие*: результатом соединения списка ***L1*** и списка ***L2*** является список ***L3***; соединение выполняется следующим образом: голова списка ***L1*** добавляется к хвосту списка ***L3***.

Ниже приведена программа, задающая заданное отношение:

append([], L, L).

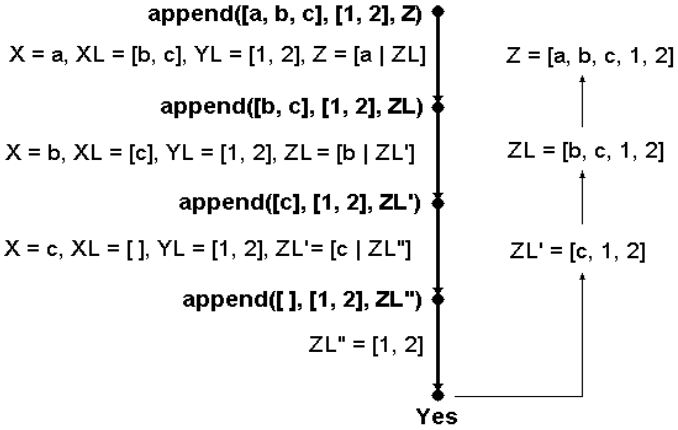
append([X | XL], YL, [X | ZL]):- append(XL, YL, ZL).

Ответом на вопрос:

?- **append([a, b, c], [1, 2], Z).**

будет **Z = [a, b, c, 1, 2].**

Графическое представление получения результата приведено на рис. 9.6.



Рим. 9.6

Здесь процесс получения результата условно разбит на два этапа – разбиение и вычисление. На первом этапе ПРОЛОГ-система ищет принципиальную возможность выполнить поставленную задачу, формируя при этом вспомогательные переменные. На втором этапе, используя вспомогательные переменные, система формирует непосредственно сам результат. Первый этап на рисунке 9.6 представлен жирными линиями, второй – более тонкими.

Отношение ***append*** может быть использовано и для выполнения операции обратной соединению двух списков, а именно для разбиения списка на возможные составляющие. Рассмотрим вопрос

?- **append(A, B, [a, b, c]).**

Начальная резольвента – ***append(A, B, [a, b, c]).*** В программе выбирается первое предложение и делается попытка унифицировать его с резольвентой. Попытка удачна, переменной ***A*** присваивается значение **[]**, а переменной ***B*** – ***[a, b, c]***. Резольвента при этом пуста, найдено первое решение. Однако начальная резольвента ***append(A, B, [a, b, c])*** может быть унифицирована и со вторым предложением программы. После унификации ***X = a A = [a | XL], B = YL, ZL = [b, c].*** Новая резольвента – ***append(XL, YL, [b, c])*** – успешно унифицируется с первым предложением программы, переменной ***XL*** присваивается значение **[]**, а переменной ***YL*** – ***[b, c]***. Резольвента пуста, найдено второе решение, при этом ***A = [a | XL] = [a], B = YL = [b, c].*** Резольвента ***append(XL, YL, [b, c])*** может быть унифицирована и со вторым предложением программы. После унификации ***X = b, XL = [b | XL′], YL = YL′, ZL′ = [c].*** Новая резольвента – ***append(XL′, YL′, [c])*** – успеш-

но унифицируется с первым предложением программы, переменной ***XL′*** присваивается значение **[]**, а переменной ***YL′*** – ***[c]***. Резольвента пуста, найдено третье решение, при этом ***A = [a | XL] = [a, b], B = YL =YL′ = =[c].*** Резольвента ***append(XL′, YL′, [c])*** успешно унифицируется со вторым предложением программы после унификации ***X = c, XL′ = [c | XL″], YL′ = =YL″, ZL″ =*** **[].** Новая резольвента – ***append(XL″, YL″,*** **[])** – успешно унифицируется только с первым предложением программы, после унификации ***XL″ = YL″ =*** **[]** – это четвертое решение, при этом ***A = [a | XL] = =[a, b | XL′] = [a, b c], B = YL = YL′ = YL″ =*** **[].** Графическое представление поиска решения приведено на рис. 9.7.

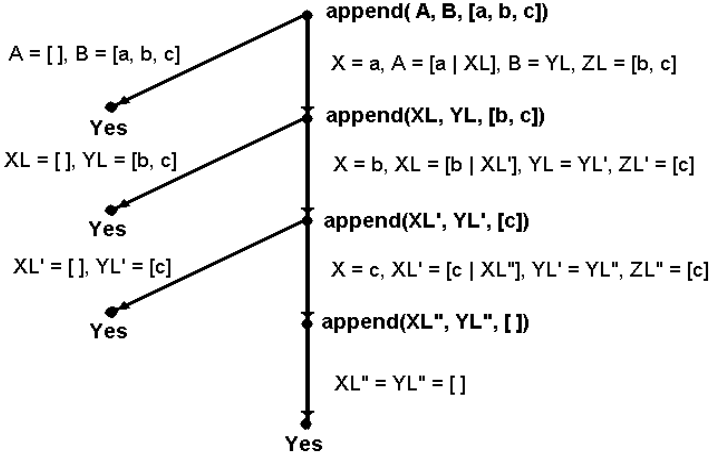


Рис. 9.7

Рассмотрим программу удаления указанного элемента из списка. Для удаления элемента из списка требуются три следующих аргумента:

- удаляемый элемент ***X***;
- список ***L1***, в который может входить ***X***;
- список ***L2***, из которого удалены все вхождения ***X***.

Предикат для удаления элемента из списка будет иметь следующий вид: **delete(X, L1, L2)**

В процессе удаления элемента из списка возможны следующие ситуации:

- X является удаляемым элементом,
- X не является удаляемым элементом.
- список пуст.

Граничное условие: из пустого списка не удаляется ничего, результатом будет также пустой список.

Рекурсивное условие:

1) если голова списка ***L1*** совпадает с удаляемым элементом ***X,*** то результатом будет список **L2**, полученный рекурсивным удалением ***X*** из ***L1***;

2) если удаляемый элемент ***X*** отличен от головы ***Z*** списка ***L1***, то результатом будет список ***L2***, голова которого ***Z***, а хвост получен рекурсивным удалением элемента ***X***.

Полная программа имеет вид:

```
delete (_, [], []).
delete (X, [X | XL], YL):- delete (X, XL, YL).
delete (X, [Z | XL], [Z | YL]):- X<>Z, delete (X, XL, YL).
```

Используя встроенный предикат «отсечение», который подробно будет описан ниже, программу удаления всех вхождений элемента в список можно представить и так:

```
delete (_, [], []).
delete (X, [X | XL], YL):- !, delete (X, XL, YL).
delete (X, [Z | XL], [Z | YL]):- delete (X, XL, YL).
```

Рассмотрим программу нахождения последнего элемента списка ***last(X, XL),*** здесь ***X*** последний элемент списка ***XL***. *Граничное условие*: единственный элемент списка является последним. *Рекурсивное условие*: если в списке элемент не единственный, то продолжить поиск в списке без первого элемента.

```
last (X, [X]).
last (X,[_ | XL]):- last (X,XL).
```

Отношение ***last (X, XL)*** можно задать с помощью отношения ***append*** следующим образом: **last (X,XL):-append(_, [X], XL).**

Интересной и полезной при работе со списками является программа, осуществляющая реверс или обращение списка. Рассмотрим два варианта такой программы. Первый вариант – с использованием программы ***append,*** которая используется здесь не для объединения двух списков, а для разбиения исходного списка на два промежуточных. *Граничное условие*: реверсом пустого списка будет пустой список. *Рекурсивное условие*: реверс непустого списка выполняется следующим образом: при помощи программы ***append*** исходный список разбиваем на два, причем второй список здесь состоит из одного последнего элемента исходного списка, этот последний элемент становится головой реверсивного списка. Программа, задающая заданное отношение, приведена ниже:

```
reverse([], []).
reverse(L, [B | RL]):-  append(A, [B], L), reverse(A, RL).
```

Второй вариант программы ***revers*** обращается к отношению ***reverse1.*** Это отношение использует промежуточный список для реверсивного отображения исходного списка и не использует предикат

append. *Граничное условие*: результатом реверсивного отображения будет промежуточный список, если исходный список пуст. *Рекурсивное условие* – реверс непустого списка выполняется следующим образом: голова исходного списка переписывается в голову промежуточного списка. При первоначальном обращении к программе ***reverse1*** промежуточный список пуст.

```
reverse(L, R):- reverse1(L, [], R).
reverse1([], R, R).
reverse1([X | L], S, R):- reverse1(L, [X | S], R).
```

В обоих вариантах программы первый аргумент задает исходный список, второй – реверсивное отображение исходного списка. На вопрос **?- reverse([a, b, c, d], R).** будет дан ответ **R = [d, c, b, a].**

Программа обращения списка без предиката ***append*** эффективнее программы с использованием ***append***. В общем случае размер дерева вывода для программы ***reverse***, использующей предикат ***append***, квадратично зависит от размера исходного списка, а в программе, использующей промежуточный список, эта зависимость носит линейный характер.

9.3 Отсечение

Рассмотрим выполнение предиката, определяющего принадлежность элемента списку:

```
member(X, [X | _]).
member(X, [_ | Y]):- member(X, Y).
```

Вопрос: ?- member(a, [a, b, a, c, a]). сгенерирует три возможных решения. Графическое представление поиска решения приведено на рис. 9.8.

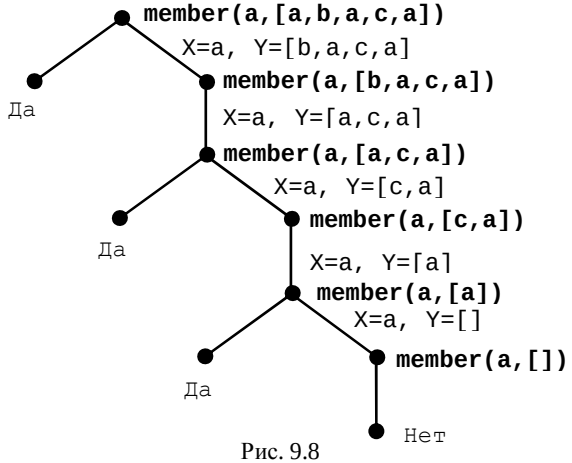


Рис. 9.8

Очевидно, если предикат используется только для определения вхождения элемента в список, то нет нужды во всех трех решениях, достаточно ответа да/нет. В ПРОЛОГе существует механизм, позволяющий отбросить ненужные решения. Механизм этот называется «отсечение». Отсечение – это встроенный предикат, затрагивающий процедурное поведение программ. Синтаксически в правиле отсечение задается как предикат «!» (CUT). Этот предикат всегда выполним, однако он влияет на процесс последующего возврата.

Можно предотвратить ненужный перебор в программе, определяющей принадлежность элемента списку, если изменить первое предложение, добавив в него отсечение следующим образом:

member(X, [X | _]):- !.
member(X, [_ | Y]):- member(X, Y).

Заданное таким образом отношение сгенерирует единственное решение.

Рассмотрим общий случай применения отсечения в предложении ***C*** процедуры ***A***.

A :- ….
C: **A :- B_1, ..., B_k, !, B_{k+1}, ..., B_n.**
A :- ….
……
A :- ….

Если текущая цель унифицирована с заголовком предложения ***C***, и цели $B_1, ..., B_k$ выполнены, то отсечение приводит к следующему результату:

1) в программе фиксируется выбор предложения ***C*** для выполнения текущей цели;

2) любые другие предложения процедуры ***A***, которые могут быть унифицированы с текущей целью, игнорируются; другими словами, все предложения процедуры, расположенные *ниже* предложения с выполненным отсечением, отбрасываются;

3) если некоторое B_i при $i>k$ не выполняется, то возврат производится не далее, чем к точке отсечения.

4) при этом все другие выборы в дереве поиска, относящиеся к вычислению цели B_i, при $i<=k$ отбрасываются, то есть отбрасываются предложения, лежащие *левее* отсечения;

5) если процесс возврата привел к точке отсечения, то отсечение становится невыполнимым, и процесс поиска возвращается к последнему выбору, сделанному перед сопоставлением текущей цели и предложения ***C***.

Эффективность программы, в которую включено отсечение, может возрасти по двум причинам:

- возрастет скорость выполнения, в силу того что будут отсутствовать попытки унификации целей, о которых заранее известно, что они приведут к неуспеху;
- уменьшится объем занимаемой памяти, так как не будут запоминаться точки возврата для последующего анализа.

Кроме того, применяя отсечение, можно повысить выразительность языка при описании взаимоисключающих правил. Однако применение отсечения приводит и к определенным проблемам, а именно, программист должен точно знать, как будут использоваться правила, содержащие отсечения.

Рассмотрим еще раз измененное определение предиката ***member***:

member(X, [X | _]):- !.

member(X, [_ | Y]):- member(X, Y).

Если предикат ***member*** используется для определения принадлежности элемента списку в вопросе **?- member(a, [a, b, a, c, a]).,** то применение отсечения вполне оправдано. Однако предикат ***member*** может использоваться не только для проверки, но и для генерации элементов из приведенного списка. Рассмотрим вопрос **?- member(Z, [a, b, c, d]).** Процедура с отсечением даст единственный ответ **Z = a,** в то время как процедура без отсечения сгенерирует следующие четыре ответа:

Z = a,
Z = b,
Z = c,
Z = d.

Таким образом, если меняется характер использования процедуры, то необходимо провести анализ применения отсечения.

Отсечение используется в следующих трех случаях:

1) когда необходимо указать ПРОЛОГ-системе, что найдено нужное правило для выполнения;

2) когда необходимо указать ПРОЛОГ-системе на немедленное прекращение поиска решений без попытки найти альтернативные решения;

3) когда необходимо указать ПРОЛОГ-системе закончить порождение альтернативных решений.

9.4 Метод «образовать и проверить»

Суть данного метода состоит в том, что один процесс или программа генерирует множество предполагаемых решений задач, а другой процесс или программа проверяет предполагаемые решения, пытаясь найти те из них, которые действительно являются решениями задачи.

Логические программы, реализующие метод «образовать и проверить», обычно содержат конъюнкцию двух целей, одна из которых действует как генератор предполагаемых решений, а вторая проверяет, являются ли эти решения приемлемыми:

find(X):- generate(X), test(X).

Если при решении вопроса **?- find(X).** успешно выполняется цель ***generate (X)*** с выдачей некоторого ***X***, то затем выполняется проверка ***test(X)***. Если проверка завершилась отказом, то производится возвращение к цели ***generate (X)***, с помощью которой генерируется следующий элемент. Процесс продолжается итерационно до тех пор, пока при успешной проверке не будет найдено решение с характерными свойствами или генератор не исчерпает все альтернативные решения. Часто в качестве генератора используется программа для предиката ***member***, порождающая множество решений.

Рассмотрим пример программы проверки существования в двух списках хотя бы одного общего элемента:

intersect(XL, YL):- member(X, XL), member(X, YL).

Первая цель ***member*** в теле этого предложения генерирует элементы из первого списка, а с помощью второй цели ***member*** проверяется, входят ли эти элементы во второй список.

Для реализации метода «образовать и проверить» часто используется процедура ***append,*** которую можно применить не только для конкатенации двух списков, но и для разбиения заданного списка на две части. Рассмотрим типичную задачу идентификации цепочек символов, в которой необходимо определить вхождение некоторой подцепочки (образа) в заданную цепочку символов (контекст). Цепочки будем представлять в виде одноуровневых списков. Пусть *s* – подцепочка-образ, *t* – цепочка-контекст, тогда вхождение *s* в *t* можно представить как конкатенацию *t = pref + s + suff*, где *pref* и *suff* – префикс и суффикс цепочки соответственно. Таким образом, определить вхождение цепочки *s* в цепочку *t* можно двумя способами:

- разбить *t* на префикс и некоторую подцепочку и выделить в этой подцепочке *s*;
- разбить *t* на некоторую подцепочку и суффикс и выделить в этой подцепочке *s*.

Ниже приведена процедура ***sublist(Slist, List)***, реализующая первый способ идентификации, здесь ***Slist*** – список-образ**,** ***List*** – список-контекст.

```
sublist(Slist, List):-   append(_, L, List),
                         append(Slist, _, L).
```

Первая цель ***append*** в теле этого предложения генерирует из списка-контекста ***List*** подсписок-префикс, обозначенный анонимной переменной, и список ***L,*** в который предположительно входит список-образ; а с помощью второй цели ***append*** проверяется возможность разбиения списка ***L*** на список-образ ***Slist*** и подсписок-суффикс, обозначенный анонимной переменной.

Процедура, реализующая второй способ идентификации, очень похожа на процедуру, реализующую первый способ идентификации.

```
sublist(Slist,List):-   append(L, _, List),
                        append(_, Slist, L).
```

Первая цель ***append*** в теле этого предложения генерирует из списка-контекста список ***L*** и подсписок-суффикс, обозначенный анонимной переменной; а с помощью второй цели ***append*** проверяется возможность разбиения списка ***L*** на список-образ и подсписок-префикс, обозначенный анонимной переменной.

На вопрос

```
?- sublist([т,о,м,а,т],[а,в,т,о,м,а,т,ы]).
```

будет дан ответ «да», а на вопрос

```
?- sublist([а,р,о,м,а,т],[а,в,т,о,м,а,т,ы]).
```

ПРОЛОГ-система ответит «нет».

Головоломки. В конкурсе эрудитов участвовали пять школьников, им показали пять портретов известных личностей и предложили назвать их. Портреты были пронумерованы в порядке их предъявления. Ниже приведены ответы:

1-й школьник: 1) Наполеон, 2) Бетховен, 3) Ньютон, 4) Шекспир, 5) Рембрандт;

2-й школьник: 1) Шекспир, 2) Рембрандт, 3) Наполеон, 4) Ньютон, 5) Бетховен;

3-й школьник: 1) Бетховен, 2) Шекспир, 3) Ньютон, 4) Рембрандт, 5) Наполеон;

4-й школьник: 1) Шекспир, 2) Бетховен, 3) Наполеон, 4) Ньютон, 5) Рембрандт;

5-й школьник: 1) Шекспир, 2) Рембрандт, 3) Ньютон, 4) Наполеон, 5) Бетховен.

Количество правильных ответов было у всех эрудитов разное, и лишь победитель назвал всех знаменитостей верно. Определить его. Программа решения данной задачи представлена ниже.

```
ans([[Наполеон, Бетховен , Ньютон, Шекспир, Рембрандт],
     [Шекспир, Рембрандт, Наполеон, Ньютон, Бетховен],
     [Бетховен, Шекспир, Ньютон, Рембрандт, Наполеон],
     [Шекспир, Бетховен, Наполеон, Ньютон, Рембрандт],
```

```
    [Шекспир, Рембрандт, Ньютон, Наполеон, Бетховен]]).
eq(0, [], []).
eq(N, [X | L1], [X | L2]):- eq(N1, L1, L2), N=N1+1, !.
eq(N, [_ | L1], [_ | L2]):- eq(N, L1, L2).

equal(_, [], []).
equal(X, [Y | LA],[N | LN]):- eq(N, X, Y), equal(X, LA, LN),
not(member(N, LN)).

solv(X):-ans(L), member(X, L), equal(X,  L, Z).
```

Отношение ***solv(X)*** находит правильный ответ из пяти представленных отношением ***ans(L).*** Отношение ***member(X, L)*** поочередно выбирает ответы эрудитов из общего списка, а каждый из выбранных ответов проверяется в процедуре ***equal(X, L, Z)*** на ограничения, сформулированные в условии задачи, а условие здесь одно: число правильных ответов у всех эрудитов должно быть разным. Непосредственно эту проверку осуществляет процедура **eq(N, L1, L2).**

```
?- solv(X).
X = [Шекспир, Бетховен, Наполеон, Ньютон, Рембрандт].
```

9.5 Циклы и повторения

В этом разделе рассмотрим способы программирования повторяющихся операций на примерах поиска в базе данных. Общий вид процедуры, выполняющей повторение, имеет следующий вид:

```
proc_repeat:- «повторяемые операции», fail.
proc_repeat.
```

В первом предложении данной процедуры конструкция ***«повторяемые операции»*** содержит несколько предикатов, как определенных в программе, так и системных. Встроенный предикат ***fail*** после успешного выполнения конструкции ***«повторяемые операции»*** вызывает возврат назад и реализует повторное выполнение указанной конструкции. В случае неудачного выполнения конструкции ***«повторяемые операции»*** выполняется второе предложение процедуры, всегда приводящее к успеху.

Вернемся к базе читаемых курсов:

```
lecture(«Ходашинский», «Искусственный интеллект»).
lecture(«Адуева», «Программирование»).
lecture(«Хон», «Банки данных»).
lecture(«Лепихина», «АИС»).
lecture («Иванова», «Физика»).
```

lecture («Байбакова», «Высшая математика»).

Пусть необходимо напечатать все изучаемые предметы, находящиеся в базе. Процедура, решающая эту задачу, приведена ниже.

write_bd:- lecture(_, X), write(X), nl, fail.
write_bd.

На вопрос **?- write_bd.** будет дан следующий ответ:

«Искусственный интеллект»
«Программирование»
«Банки данных»
«АИС»
«Физика»
«Высшая математика»

Рассмотрим процесс получения данного ответа:

1) формируется резольвента ***write_bd***;

2) выбирается первое предложение в процедуре ***write_bd,*** формируя новую резольвенту ***lecture(_, X), write(X), nl, fail***;

3) выбирается первая цель в резольвенте – ***lecture(_, X)***;

4) просматривается база данных в попытке унифицировать выбранную цель и факт из базы, попытка оказывается успешной, в базе найден первый факт ***lecture(«Ходашинский», «Искусственный интеллект»)***. Переменная ***X*** с этого момента конкретизирована и ей соответствует значение «Искусственный интеллект»;

5) существуют и другие факты, которые могут быть унифицированы с целью ***lecture(_, X),*** поэтому ПРОЛОГ-система сохраняет возможность вернуться к оставшимся фактам;

6) первая цель резольвенты выполнена, система переходит к следующим целям – ***write(«Искусственный интеллект»)*** и ***nl***;

7) предикат **fail** возвращает процесс вычисления к целям ***write(X)*** и ***nl,*** это встроенные предикаты, поэтому возврат идет до цели ***lecture(_, X)***;

8) выбирается следующий факт из оставшихся – ***lecture-(«Адуева», «Программирование»)***.

Процесс повторяется до тех пор, пока не будет просмотрена вся база данных, то есть цель ***lecture(_, X)*** не может быть унифицирована, тогда выполняется второе предложение процедуры ***write_bd,*** и вычисление успешно завершается.

Рассмотрим вариант цикла, основанного на незавершенности предиката ***repeat***, который задается следующим образом:

repeat.
repeat:-repeat.

Рассмотрим программу, считывающую данные из входного потока и выводящую эти данные в выходной поток до тех пор, пока не будет введена строка ***stop***.

rewrite:- repeat, read (X), check(X), !.
check("stop"):-!.
check(X):- write(X), nl, fail.

В данной программе решение цели ***check(X)*** приводит к безуспешному вычислению, если только ***X*** не строка ***"stop"***. Безуспешное вычисление вызывает возврат к цели ***repeat***, цель выполняется, вновь считывается строка и выводится в выходной поток. Отсечение в определении предиката ***rewrite*** гарантирует от позднейшего повторения циклов ***repeat***. Repeat-циклы применяются в ПРОЛОГе для описания интерактивного взаимодействия с внешней средой путем повторяющегося ввода и/или вывода. В repeat-цикле обязательно должен присутствовать предикат, гарантированно приводящий к безуспешным вычислениям. В нашем случае это цель ***check(X)***. Полезное эвристическое правило построения repeat-циклов: в теле правила, содержащего цель ***repeat***, должно быть отсечение, предохраняющее от незавершающихся вычислений при возврате в цикл. Практическая ценность repeat-циклов заключается в том, что они благодаря использованию ***fail*** при больших вычислениях не забивают стек. Так, процедура ***rewrite*** могла быть определена иначе, например,

rewrite:- read (X), X<>”stop”, write(X), rewrite.

Здесь повторяющиеся операции задаются непосредственно повторным обращением к процедуре, а это приведет через определенное число обращений к переполнению стека.

Контрольные вопросы

1. В чем заключается принципиальное отличие итерации от рекурсии?
2. Напишите ПРОЛОГ-процедуру, удаляющую элемент из двухуровневого списка.
3. Когда и почему в логическом программировании применяется отсечение?
4. Напишите ПРОЛОГ-программу размещения четырех ферзей на доске 4×4 так, чтобы обеспечить отсутствие взаимных нападений.
5. Какие стилистические соглашения, по Вашему мнению, могут улучшить программирование на языке ПРОЛОГ?

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО КУРСУ «МЕТОДЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА, БАЗЫ ЗНАНИЙ, ЭКСПЕРТНЫЕ СИСТЕМЫ»

ВВЕДЕНИЕ

Задачей искусственного интеллекта как научного направления является воссоздание с помощью компьютера разумных рассуждений и действий. Из всего многообразия научных и технических исследований, называемых искусственным интеллектом, в учебном курсе «Методы искусственного интеллекта, базы знаний, экспертные системы» выбраны аспекты, связанные с проблемами представления знаний и вывода на знаниях, а также некоторые вопросы построения экспертных систем, являющихся одним из классов интеллектуальных систем.

Цель курса – ознакомление студентов с методами и моделями искусственного интеллекта.

После изучения курса студент должен:

***иметь представление*:**

- о знаниях, методах их получения, хранения и обработки;
- об искусственном интеллекте как научном направлении и о решаемых здесь задачах;
- о возможностях технологии экспертных систем и путях применения данных технологий в различных областях;

***знать*:**

- основные модели и методы искусственного интеллекта;
- принципы построения и методы разработки экспертных систем;

владеть языком программирования ПРОЛОГ как средством разработки интеллектуальных систем.

Для изучения курса студентам необходимо усвоить следующие дисциплины и темы:

- общую математику: алгебру; вероятность и статистику;
- специальные главы математики: логику;
- информатику: алгоритмизацию и программирование.

1 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ТЕМАМ КУРСА

Тема 1. Предмет и методы дисциплины.

История становления.
Терминология.
Знание. Модели представления знаний.

Методические указания

Искусственный интеллект – научное направление, появившееся во второй половине двадцатого века на стыке таких научных дисциплин как математика, вычислительная техника, логика, программирование, психология, лингвистика, нейрофизиология. Задачей этого научного направления является воссоздание с помощью компьютера разумных рассуждений и действий.

Раскройте содержание понятия «интеллект».

Изучая историю развития дисциплины «Искусственный интеллект», обратите внимание на смену парадигм.

Определите на множестве компьютерных систем место интеллектуальных систем; систем, основанных на знаниях; экспертных систем.

Объектом изучения дисциплины «Искусственный интеллект» являются процедуры, используемые при решении человеком интеллектуальных задач с целью воссоздания компьютерных моделей этих процедур. Основные методы связаны с применением разного рода программных и теоретических моделей. Обратите внимание на то, что в когнитивистском и бионическом направлениях эти методы различны.

Изучите особенности, присущие форме представления информации в компьютере, которую называют «знание».

Обратите внимание на противоречивость таких требований к представлению знаний, как однородность и простота понимания.

Рассмотрите четыре типичные модели представления знаний: логические, продукционные, фреймовые и сетевые модели.

Контрольные вопросы

1. Что изучает дисциплина «Искусственный интеллект»?
2. Чем отличаются алгоритмический и эвристический подходы решению задач?
3. Какие особенности присущи области применения искусственного интеллекта?
4. В чем заключается принципиальное отличие знаний и данных?
5. На каких формальных теориях построены модели представления знаний?

Тема 2. Логическая модель.

Логика высказываний.
Логика предикатов.

Методические указания

Ознакомьтесь с основными конструкциями логики высказываний. Уясните правила построения формул в логике высказываний.

Обратите внимание на сходство и отличия в применении логических связок в естественном языке и в логических формулах. Приведите свои примеры сложных высказываний на естественном языке и переведите их в логические формулы.

Изучите бесскобочную форму записи логических формул.

Раскройте содержание понятия «интерпретация формулы логики высказываний», а также понятий «общезначимость», «противоречивость», «необщезначимость», «непротиворечивость», «выполнимость» формул логики высказываний. Изучите эти понятия на своих собственных примерах.

Ознакомьтесь с правилами эквивалентных преобразований формул. Обратите внимание на то, что эквивалентные преобразования действуют в обоих направлениях.

Разберитесь с понятием логического следствия. Выясните, как связаны между собой логическое следствие, общезначимость и противоречивость.

Изучите способы логического вывода в логике высказываний с точки зрения реализации их на компьютере.

Разберитесь с синтаксисом логики предикатов. Уясните основные синтаксические отличия логики высказываний и логики предикатов.

Раскройте содержание понятия «интерпретация формулы логики предикатов первого порядка», обратите внимание на отличия в правилах интерпретации формул логики предикатов и формул логики высказываний.

Изучите способ получения универсальной области интерпретации формул логики предикатов. Разберитесь с тем, как может быть использована теорема Эрбрана в логическом выводе.

Уясните принципиальные отличия в использовании теоремы Эрбрана и метода резолюций в логическом выводе.

Контрольные вопросы

1. Какие допущения приняты при описании мира с помощью логических моделей?

2. Что такое правильно построенная формула в логике высказываний и в логике предикатов?

3. Могут ли быть использованы способы логического вывода, определенные в логике высказываний, для вывода в логике предикатов? Если «да», то опишите, как это можно сделать; если «нет», то объясните, почему невозможно.

Тема 3. Продукционная модель.

Представление.
Вывод.
Система OPS5.

Методические указания

Рассмотрите варианты задания продукционного правила. Обратите внимание на сходство и отличие в определении правил.

Изучите структуру продукционной системы и способы получения вывода в системе. Выясните, какие функции выполняет каждый компонент системы.

Рассмотрите два способа вывода в продукционных системах: прямой и обратный. Придумайте свой пример продукционной системы, разберите прямой и обратный вывод в данной системе.

Ознакомьтесь со способами визуального представления правил в продукционных системах. Представьте правила Вашей продукционной системы в виде И/ИЛИ-графа.

Раскройте содержание понятия «конфликтный набор». Изучите способы разрешения конфликтов в продукционных системах в зависимости от типа вывода. Приведите примеры.

Рассмотрите особенности архитектуры продукционной системы, использующей для вывода модель доски объявлений.

Назовите сильные и слабые стороны продукционных систем.

Изучите типы данных в OPS5 и синтаксис задания правила. Приведите свои примеры.

Предложите свой язык разработки продукционных систем. Обоснуйте выбранный Вами подход к определению продукционных правил и данных.

Контрольные вопросы

1. Что такое продукционная система?
2. Каковы функции рабочей памяти и машины логического вывода?
3. Что такое цепочка логического вывода?

4. Обоснуйте преимущества и недостатки прямого и обратного вывода.

5. Что такое граф типа И/ИЛИ, для каких целей используются такие графы в продукционных системах?

6. Какие типы выводов возможны в продукционных системах?

7. Как задаются и как используются переменные в OPS5?

Тема 4. Фреймовая модель

Представление.
Выводы во фреймовых системах.

Методические указания

Изучите структуру фрейма. Обратите внимание на процедурно-декларативный характер описания информации во фреймовых представлениях.

Разберитесь со структурой данных фрейма в языке FRL. Обратите внимание на использование системных имен во фреймах, на связь значения слота с типом данных и условием наследования.

Уясните условия запуска демонов и присоединенных процедур.

Выясните способы вывода во фреймовых системах. Разберитесь, на свойствах каких отношений основан механизм наследования, а также как пишутся демоны и служебные процедуры.

Выявите сходства и отличия фреймовых и объектно-ориентированных языков программирования.

Контрольные вопросы

1. Какой тип информации представлен во фрейме?
2. Что такое слот фрейма, как он определяется во фрейме?
3. Каковы функции ISA-отношения во фреймовых системах?
4. Назовите проблемы, которые встречаются при выводе во фреймовых системах, и способы решения этих проблем.

Тема 5. Семантические сети.

Модели семантических сетей.
Выводы в семантических сетях.

Методические указания

Раскройте содержание понятия «семантика». Дайте определение синтаксическим, семантическим и прагматическим отношениям, принятым в семиотике.

Разберитесь с тем, что принято называть семантической сетью. Обратите внимание на целостность образа сети и неразделимость синтаксических и семантико-прагматических знаний о внешнем мире.

Изучите TLC-модель. Назовите основные отношения, принятые в данной модели. Определите достоинства и недостатки TLC-модели.

Разберитесь с применением первичного и вторичного определения понятия при представлении разных видов информации в виде понятийных структур.

Рассмотрите средства, дающие возможность представлять в сети события и действия. Обратите внимание на два основных уровня языка, принятых в лингвистике: уровень поверхностных структур и уровень глубинных структур. Приведите пример.

Изучите способы вывода в семантических сетях. Выясните, на каких отношениях определен механизм наследования.

Уясните механизм вывода в семантических сетях, основанный на построении подсети, соответствующей вопросу, и сопоставлении ее с базой знаний. Разберитесь со способом вывода, называемым перекрестным поиском.

Рассмотрите семантическую сеть специального вида, носящую название «функциональная семантическая сеть». Обратите внимание на то, как задаются параметры, участвующие в решении задачи, и как задаются функциональные отношения, связывающие между собой эти параметры.

Разберитесь с механизмами вывода в функциональной семантической сети, основанными на распространяющихся волнах и паросочетаниях. Выявите достоинства и недостатки рассмотренных методов.

Контрольные вопросы

1. Определите основную концепцию представления знаний на основе семантической сети.
2. Что показывают вершины и дуги в семантической сети общего типа?
3. Что такое падеж Филмора? Какую информацию можно представить с помощью таких падежей?
4. На каких отношениях определено наследование атрибутов и свойств сети?
5. Что показывают вершины и дуги в функциональной семантической сети?
6. Как определены решения в функциональной семантической сети?
7. Когда прекращается распространение волн в функциональной семантической сети?

Тема 6. Нечеткие знания.

Нечеткие множества.
Операции на нечетких множествах.
Нечеткие отношения.
Вывод на нечетких знаниях.
Ненадежные знания.

Методические указания

Выясните, как задается нечеткое множество и чем нечеткое множество отличается от четкого. Обратите внимание на обозначения в случае непрерывного и дискретного множества.

Изучите операции на нечетких множествах. Дайте графическую интерпретацию указанных операций.

Выясните, как задается нечеткое отношение. Изучите задание отношений в случае конечных множеств с помощью матрицы отношений и взвешенного графа. Обратите внимание на определение операции свертки max-min двух нечетких множеств.

Разберитесь с методом приближенных рассуждений, в котором посылки являются нечеткими понятиями. Уясните, как получаются нечеткие отношения из нечеткого условного высказывания. Сравните методы нечетких рассуждений, использующие перечисленные типы нечетких отношений в правилах обобщенного модус поненс.

Выясните, как можно использовать метод разбиения сложных задач на подзадачи с использованием дерева И-ИЛИ-КОМБ. Обратите внимание на способ вычисления степени надежности при связи КОМБ. Изучите метод MYCIN.

Контрольные вопросы

1. Какова основная идея, лежащая в основе понятия «нечеткое множество»?
2. Для каких целей используются нечеткие отношения?
3. В чем заключается принципиальное отличие традиционного правила модус поненс от обобщенного?
4. Как может быть использовано обобщенное правило модус толленс для нечетких рассуждений?
5. Зачем при разбиении сложных задач вводится связь КОМБ?
6. Возможно ли в общем случае устранение нечеткости и ненадежности при разработке интеллектуальных систем?

Тема 7. Основы построения экспертных систем.

Структура и разработчики экспертных систем.
Основные функции экспертных систем.
Этапы разработки экспертных систем.
Стадии разработки системы.
Инструментальные средства разработки.
Средства объяснения.
Приобретение знаний.

Методические указания

Уясните, к какому классу относятся экспертные системы. Выясните, какие нововведения делают экспертную систему эффективной, ценной и имеют наибольшее потенциальное значение.

Изучите особенности архитектуры экспертной системы. Обратите внимание на компоненты, присущие только экспертным системам.

Процесс построения экспертных систем называют инженерией знаний. Разберитесь с тем, кто участвует в проектировании экспертных систем, выясните их функции и способы взаимодействия.

Выявите типичные задачи, выполняемые экспертными системами. Обратите внимание на характерные особенности, присущие этим задачам: большое пространство решений, условное рассуждение, применение зашумленных данных.

Изучите технологию проектирования и разработки экспертных систем. Выясните, какие задачи решаются на каждом из этапов разработки. Сравните технологию проектирования экспертных систем с технологией проектирования традиционных программных систем.

Выясните, что представляют собой инструментальные средства, облегчающие создание экспертных систем. Проведите сравнительный анализ средств, используемых для построения экспертных систем.

Выясните, чем вызвана важность объяснительного компонента в экспертных системах. Рассмотрите классификацию типов объяснения.

Рассмотрите методологические и технологические проблемы приобретения знаний. Изучите фазы и модели приобретения знаний.

Контрольные вопросы

1. Назовите те характеристики экспертных систем, которые отличают их от обычных программ.

2. Возможно ли применение статистических методов в экспертных системах? Почему?

3. Какие задачи решаются на каждом из этапов разработки экспертных систем?

4. Перечислите и охарактеризуйте стадии существования экспертных систем.

Тема 8. Язык программирования ПРОЛОГ.

Простейшие программы.
Термы.
Переменные и константы.
Сложные термы.
Поиск решения.

Методические указания

Изучите основные конструкции логического программирования. Выясните, как эти конструкции оформляются синтаксически. Обратите внимание на то, как объявляются факты об объектах, как задаются вопросы относительно известных фактов, как представляются отношения в виде правил.

Уясните понятие «процедура». Обратите внимание на то, какими логическими операциями связаны правила и цели в процедуре.

Рассмотрите представление ПРОЛОГ-программы и поиск ответа на вопрос на *И-ИЛИ*-дереве. Выясните, как строится дерево поиска ответа на вопрос и что такое «бектрекинг». Изучите обход дерева при поиске ответа на простой и конъюнктивный вопрос.

Разберитесь с типами данных в ПРОЛОГе и с основными операциями над термами ПРОЛОГа. Рассмотрите вхождения переменных в ПРОЛОГ-правило. Обратите внимание на отличия в использовании переменных в логических и традиционных языках программирования.

Изучите поиск решения в логической программе. Обратите внимание на операции конкретизации и унификации.

Контрольные вопросы

1. Чем принципиально отличаются языки логического программирования от традиционных языков программирования?
2. Какую роль играет переменная в логическом языке программирования? Какова область ее действия?
3. Как задается сложный терм?
4. Что такое цель в логическом программировании?
5. Как задается и выполняется ПРОЛОГ-программа? Перечислите типы ответов на вопросы.
6. Опишите свое генеалогическое древо средствами языка ПРОЛОГ. Рассмотрите действия ПРОЛОГ-системы при ответе на вопрос к базе данных, описывающей данное генеалогическое древо.

Тема 9. Техника ПРОЛОГ-программирования

Рекурсия и итерация.
Списки.
Отсечение.
Метод «образовать и проверить».
Циклы и повторения.

Методические указания

Разберитесь с понятиями «рекурсия» и «итерация». Обратите внимание на то, как задаются рекурсивное и итеративное определения процедуры. Рассмотрите этапы нахождения ПРОЛОГ-системой ответа на поставленный запрос при выполнении рекурсивных и итеративных процедур.

Рассмотрите способы задания списковых структур в ПРОЛОГе. Обратите внимание на упорядоченность данной структуры и на то, что пустой список не имеет ни головы, ни хвоста. Изучите основные операции на списках.

Разберитесь со способом управления работой механизма возврата, называемым отсечением. Выясните, к какому результату приводит отсечение при выполнении ПРОЛОГ-программы. Выявите причины возрастания эффективности программы, в которую включено отсечение. Обратите внимание на возможные нежелательные эффекты при применении отсечения.

Выявите суть метода «образовать и проверить». Рассмотрите программы, часто используемые для генерации и проверки решений.

Рассмотрите способы программирования повторяющихся операций в языке ПРОЛОГ. Обратите внимание на то, когда и как могут быть применены repeat-циклы.

Сформулируйте общие принципы хорошего программирования на языке ПРОЛОГ.

Контрольные вопросы

1. Чем определяется важность рекурсии в логическом программировании?
2. В чем заключается принципиальное отличие итерации от рекурсии?
3. Напишите ПРОЛОГ-процедуру, удаляющую элемент из двухуровневого списка.
4. Когда и почему в логическом программировании применяется отсечение?
5. Напишите ПРОЛОГ-программу размещения четырех ферзей на доске 4×4 так, чтобы обеспечить отсутствие взаимных нападений.
6. Какие стилистические соглашения по Вашему мнению могут улучшить программирование на языке ПРОЛОГ?

2 КОНТРОЛЬНЫЕ РАБОТЫ

2.1 Контрольная работа № 1

2.1.1 Методические указания к контрольной работе № 1

Для выполнения задания 1 необходимо в тексте задания выделить простые предложения, обозначить их как атомы и затем представить каждое предложение в виде формулы. Например,

если солнце село в тучу (С), то завтра будет дождь (Д);
солнце село в тучу;
следовательно, завтра будет дождь;
доказать.

Две посылки и заключение будут представлены следующим образом:

$С \to Д$
С

Д

Способ 1. Воспользуемся ОПРЕДЕЛЕНИЕМ логического следствия и таблицами истинности.

С	Д	$С \to Д$	$(С \to Д) \wedge С$
И	**И**	**И**	**И**
И	**Л**	**Л**	**Л**
Л	**И**	**И**	**Л**
Л	**Л**	**И**	**Л**

Конъюнкция посылок истинна только при одной интерпретации, заданной первой строкой, в этой же интерпретации истинным является и заключение (Д), следовательно, Д является логическим следствием посылок $(С \to Д)$ и (С).

Способ 2. Воспользуемся ТЕОРЕМОЙ 1 и таблицами истинности.

С	Д	$С \to Д$	$(С \to Д) \wedge С$	$((С \to Д) \wedge С) \to Д$
И	**И**	**И**	**И**	**И**
И	**Л**	**Л**	**Л**	**И**
Л	**И**	**И**	**Л**	**И**
Л	**Л**	**И**	**Л**	**И**

Так как формула ((С → Д) ∧С) → Д общезначима, то Д является логическим следствием посылок (С → Д) и (С).

Способ 3. Воспользуемся ТЕОРЕМОЙ 2 и таблицами истинности.

С	Д	~Д	С → Д	(С → Д) ∧С	(С → Д) ∧С ∧~Д
И	**И**	**Л**	**И**	**И**	**Л**
И	**Л**	**И**	**Л**	**Л**	**Л**
Л	**И**	**Л**	**И**	**Л**	**Л**
Л	**Л**	**И**	**И**	**Л**	**Л**

Так как формула (С → Д) ∧С ∧~Д противоречива, то Д является логическим следствием посылок (С → Д) и (С).

Способ 4. Воспользуемся ТЕОРЕМОЙ 1 и эквивалентными преобразованиями формул.

((С → Д) ∧С) → Д = ((~С ∨ Д) ∧С) → Д = ~((~С ∨ Д) ∧С) ∨ Д =

=((С ∧ ~Д) ∨ ~С) ∨ Д =(С ∧ ~Д) ∨ ~С ∨ Д =(С ∨ ~С ∨ Д) ∧(~С ∨ Д ∨ ~Д) =

=(■∨ Д) ∧(~С ∨ ■) = (■ ∧ ■) = ■,

здесь символ ■ означает общезначимую формулу. Так как в результате эквивалентных преобразований получена общезначимая формула, то Д является логическим следствием посылок (С → Д) и (С).

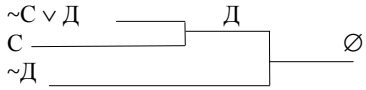
Способ 5. Воспользуемся ТЕОРЕМОЙ 2 и эквивалентными преобразованиями формул.

((С → Д) ∧С) ∧~ Д = ((~С ∨ Д) ∧С) ∧~ Д = (~С ∨ Д) ∧С ∧~ Д =

=(~С ∧С ∧~ Д)∨(Д ∧С ∧~ Д) =

=(□ ∧~ Д)∨(С ∧ □) = □∨ □= □, здесь символ □ означает противоречивую формулу. Так как в результате эквивалентных преобразований получена противоречивая формула, то Д является логическим следствием посылок (С → Д) и (С).

Способ 6. Воспользуемся методом резолюций. Доказывать будем невыполнимость множества дизъюнктов, для этого необходимо, согласно ТЕОРЕМЕ 2, взять отрицание заключения.



Так как в процессе резолютивного вывода получен пустой дизъюнкт, то Д является логическим следствием посылок (С → Д) и (С).

При выполнении задания 2 следует иметь в виду, что процедура Эрбрана предполагает генерацию основных примеров дизъюнктов из заданного множества. Для генерации основных примеров необходимо

множество констант, а таким множеством является эрбрановский универсум. Если на множестве основных примеров существует хотя бы одно невыполнимое, то заданное множество дизъюнктов невыполнимо. Например, пусть S = {P(x), ~P(f(a))}, тогда эрбрановский универсум H_∞ = {a, f(a), f(f(a)), f(f(f(a))), ... }, если вместо переменной ***x*** подставить f(a), то получим невыполнимое множество основных примеров S'= {P(f(a)), ~P(f(a))}, а это, в свою очередь, означает, что множество S невыполнимо.

Для выполнения задания 3 необходимо в тексте задания выделить простые предложения, обозначить их как атомы и затем представить каждое предложение в виде формулы. Пример, рассмотрим следующие утверждения:

F1: Каждый, кто хранит деньги в банке, получает проценты.

G: Если не выплачиваются проценты, то никто не хранит деньги в банке.

Выделим и обозначим в тексте простые предложения:

S(x, y) – *x* хранит *y* в банке;

M(x) – *x* - деньги;

I(x) – *x* - проценты;

P(x, y) – *x* получает *y*.

F1: $(\forall x)((\exists y)(S(x, y) \wedge M(y)) \rightarrow (\exists z)(I(z) \wedge P(x, z)))$.

G: $\sim(\exists z)I(z) \rightarrow (\forall x)(\forall y)(S(x, y) \rightarrow \sim M(y))$.

Переведем посылку в дизъюнкты:

$(\forall x)((\exists y)(S(x, y) \wedge M(y)) \rightarrow (\exists z)(I(z) \wedge P(x, z))) = (\forall x)(\sim(\exists y)(S(x, y) \wedge \wedge M(y)) \vee (\exists z)(I(z) \wedge P(x, z))) = (\forall x)((\forall y)(\sim S(x, y) \vee \sim M(y)) \vee (\exists z)(I(z) \wedge P(x, z))) = (\forall x)(\forall y)(\exists z)(\sim S(x, y) \vee \sim M(y) \vee (I(z) \wedge P(x, z)))$;

$(\forall x)(\forall y)(\exists z)(\sim S(x, y) \vee \sim M(y) \vee (I(z) \wedge P(x, z)))$ – предваренная нормальная форма;

$(\forall x)(\forall y)(\exists z)((\sim S(x, y) \vee \sim M(y) \vee I(z)) \wedge (\sim S(x, y) \vee \sim M(y) \vee P(x, z))) =$

$=(\forall x)(\forall y)((\sim S(x, y) \vee \sim M(y) \vee I(f(x, y))) \wedge (\sim S(x, y) \vee \sim M(y) \vee P(x, f(x, y))))$ – сколемовская стандартная форма;

дизъюнкты, полученные из стандартной формы:

$\sim S(x, y) \vee \sim M(y) \vee I(f(x, y))$,

$\sim S(x, y) \vee \sim M(y) \vee P(x, f(x, y))$.

Переведем отрицание заключения в дизъюнкты,

$\sim(\sim(\exists z)I(z) \rightarrow (\forall x)(\forall y)(S(x, y) \rightarrow \sim M(y))) = \sim((\exists z)I(z) \vee (\forall x)(\forall y)(\sim S(x, y) \vee \sim M(y))) = \sim((\exists z)(\forall x)(\forall y)(I(z) \vee \sim S(x, y) \vee \sim M(y))) = (\forall z)(\exists x)(\exists y)(\sim I(z) \wedge S(x, y) \wedge M(y))$ – предваренная нормальная форма;

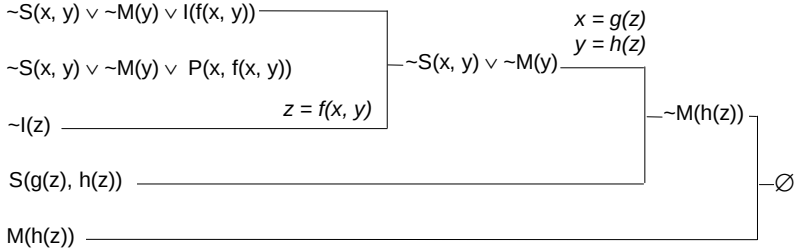
$(\forall z)(\sim I(z) \wedge S(g(z), h(z)) \wedge M(h(z)))$ – сколемовская стандартная форма;

дизъюнкты, полученные из стандартной формы:

$\sim I(z)$,

S(g(z), h(z)),
M(h(z)).

Резолютивный вывод на дизъюнктах, полученны из посылки и отрицания заключения, имеет следующий вид:



Получен пустой дизъюнкт, заключение доказано.

2.1.2 Задания для контрольной работы № 1

Вариант 1

Задание 1. Если Степан не знал о необходимости декларировать доход, то он плохой законодатель. Если он знал и не декларировал, то он мошенник. Если Степан является плохим законодателем или мошенником, то ему нет места в Думе. Степан не декларировал свой доход. Следовательно, ему нет места в Думе. Доказать всеми возможными способами.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов S = {P(x, a, g(x, b)), ~P(f(y), z, g(f(a), b))}.

Задание 3. Ни один человек не является четвероногим. Все женщины – люди. Следовательно, ни одна женщина не является четвероногой. Доказать.

Вариант 2

Задание 1. Если исход скачек будет предрешен сговором или в игорных домах будут орудовать шулеры, то доходы от туризма упадут, и город пострадает. Если доходы от туризма упадут, полиция будет довольна. Полиция никогда не бывает довольна. Следовательно, исход скачек не предрешен сговором. Доказать всеми возможными способами.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов S = {P(x), Q(x, f(x)) ∨ ~P(x), ~Q(g(y), z)}.

Задание 3. Каждый член комитета богат и демократ. Некоторые члены комитета – старики. Следовательно, существуют старики-демократы. Доказать.

Вариант 3

Задание 1. Если 6 – составное число, то 12 – составное число. Если 12 – составное число, то существует простое число, большее чем 12. Если существует простое число, большее чем 12, то существует составное число, большее, чем 12. Если 6 делится на 2, то 6 – составное число. 12 – составное число. Следовательно, 6 – составное число. Доказать всеми возможными способами.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов $S = \{P(x), \ \sim P(x) \vee Q(x, a), \sim Q(y,a)\}$.

Задание 3. Некоторые республиканцы любят всех демократов. Ни один республиканец не любит ни одного социалиста. Следовательно, ни один демократ не является социалистом. Доказать.

Вариант 4

Задание 1. Контракт будет выполнен тогда и только тогда, когда дом будет закончен в феврале. Если дом будет закончен в феврале, то мы можем переезжать 1-го марта. Если мы не можем переезжать 1-го марта, то мы должны внести квартплату за март. Если контракт не будет выполнен, то мы должны внести квартплату за март. Следовательно, мы должны внести квартплату за март. Доказать всеми возможными способами.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов $S = \{\sim P(x) \vee Q(x, f(x)), \ P(x), \sim Q(g(z), y)\}$.

Задание 3. Ни один первокурсник не любит второкурсников. Все, живущие на шестом этаже, – второкурсники. Следовательно, ни один первокурсник не любит никого из живущих на шестом этаже. Доказать.

Вариант 5

Задание 1. Если я пойду завтра на первое занятие, то должен буду встать рано, а если я пойду вечером на танцы, то лягу спать поздно. Если я лягу спать поздно, а встану рано, то я буду вынужден довольствоваться пятью часами сна. Я не могу довольствоваться пятью часами сна. Следовательно, я или не пойду завтра на первое занятие, или не пойду вечером на танцы. Доказать всеми возможными способами.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов

$$S = \{\sim C(x) \vee W(x), \sim C(x) \vee R(x), C(a) , O(a), \sim O(x) \vee \sim R(x)\}.$$

Задание 3. Ни один торговец наркотиками не является наркоманом. Некоторые наркоманы привлекались к ответственности. Следовательно, некоторые люди, привлекавшиеся к ответственности, не являются торговцами наркотиками. Доказать.

Вариант 6

Задание 1. Если Мери бросила Джона, то она уехала или в Россию, или в Израиль. Если Мери уехала в Россию, то ее арестовал КГБ. Если Мери уехала в Израиль, то ее арестовал Мосад. Мери не арестовал ни Мосад, ни КГБ. Значит Мери не бросила Джона. Доказать всеми возможными способами.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов $S = \{P(x) \vee Q(x, f(x)),\ \sim P(x), \sim Q(g(y), z)\}$.

Задание 3. Студенты суть граждане. Следовательно, голоса студентов суть голоса граждан. Доказать.

Вариант 7

Задание 1. Халиф Омар, сжегший Александрийскую библиотеку, рассуждал так: если ваши книги согласны с Кораном, то они излишни; если они не согласны с Кораном, то они вредны; но вредные или излишние книги следует уничтожать; значит, ваши книги следует уничтожить. Доказать правильность рассуждений халифа.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов

$$S = \{P(a), \sim D(y) \vee L(a, y), \sim P(x) \vee \sim Q(y) \vee \sim L(x, y), D(b)\ ,\ Q(b)\}.$$

Задание 3. Никакой торговец подержанными автомобилями не покупает подержанный автомобиль для своей семьи. Некоторые люди, покупающие подержанные автомобили для своих семей, – жулики. Следовательно, некоторые жулики не являются торговцами подержанными автомобилями. Доказать.

Вариант 8

Задание 1. Или Маша и Ваня одного возраста, или Маша старше Вани. Если Маша и Ваня одного возраста, то Наташа и Ваня не одного возраста. Если Маша старше Вани, то Ваня старше Пети. Следовательно, или Наташа и Ваня не одного возраста, или Ваня старше Пети. Доказать всеми возможными способами.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов

$$S = \{\sim S(y) \vee \sim C(y), S(b), V(a, b), \sim C(z) \vee V(a, z)\}.$$

Задание 3. Некоторые пациенты любят своих докторов. Ни один пациент не любит знахаря. Следовательно, никакой доктор не является знахарем. Доказать.

Вариант 9

Задание 1. Если я поеду автобусом, а автобус опоздает, то я пропущу назначенное свидание. Если я пропущу назначенное свидание и буду огорчен, то мне не следует ехать домой. Если я не получу эту работу, то я буду огорчен и мне следует поехать домой. Следовательно, если я поеду домой автобусом и автобус опоздает, то я получу эту работу. Доказать всеми возможными способами.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов

$S = \{\sim S(x, y) \vee \sim M(y) \vee I(f(x)), \sim S(x, y) \vee \sim M(y) \vee E(x, f(x)), \sim I(z), S(a, b), M(b)\}$.

Задание 3. Все первокурсники встречаются со всеми второкурсниками. Ни один первокурсник не встречается ни с одним студентом предпоследнего курса. Существуют первокурсники. Следовательно, ни один второкурсник не является студентом предпоследнего курса. Доказать.

Вариант 10

Задание 1. Если завтра будет холодно, я надену шубу, если рукав будет починен. Завтра будет холодно, а рукав не будет починен. Следовательно, я не надену шубу. Доказать всеми возможными способами.

Задание 2. Используя процедуру Эрбрана, доказать невыполнимость множества дизъюнктов $S = \{\sim E(x) \vee V(x) \vee S(x, f(x)), \sim E(x) \vee V(x) \vee \vee C(f(x)), P(a), E(a), \sim S(a, y) \vee P(y), \sim P(x) \vee \sim V(x), \sim P(x) \vee \sim C(x)\}$.

Задание 3. Боб – мальчик, у которого нет автомобиля. Джейн любит только тех мальчиков, у которых есть автомобили. Следовательно, Джейн не любит Боба. Доказать.

2.2 Контрольная работа № 2

2.2.1 Методические указания к контрольной работе № 2

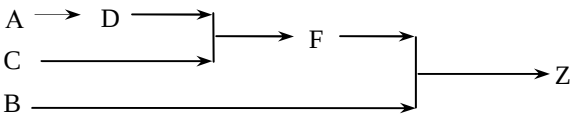
Задание 1. Пусть база правил в продукционной системе имеет содержимое:

если F и B то Z; если C и D то F; если A то D;

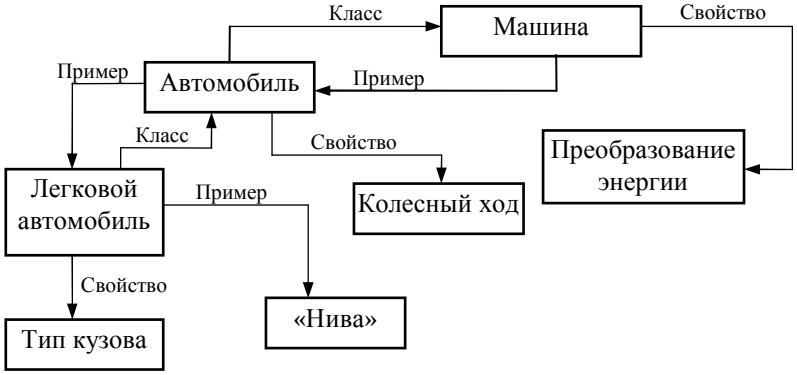
рабочая память: A, B, H, C.

Рассмотрим, каким образом «работают» правила. Система построена так, что один раз выбранное правило из базы правил выполняться будет только один раз. Оно как бы «выгорает». Первым выго-

рает правило «если A то D», так как A уже имеется в базе данных. В качестве следствия этого правила получается логический вывод о наличии ситуации D, которая заносится в рабочую область. Это вызывает выгорание правила «если C и D то F», и, как следствие, выводится ситуация F, и она заносится в базу данных. Это, в свою очередь, вызывает выгорание правила «если F и B то Z» с занесением Z в базу данных. Такой способ называется прямым выводом. Графически вывод можно изобразить следующим образом:



Задание 2. При построении TLC-модели необходимо определить взаимосвязи между понятиями. Построим TLC-модель понятия «автомобиль».



Задание 3. Пусть имеются следующие посылки:
x – не очень маленькое;
если x – маленькое, то y – большое, иначе y – маленькое.
Найти значения ***y'***. Множество U = 1+2+3.

маленькое = $^{1}/_{1} + {}^{0.4}/_{2}$; ***большое*** = $^{0.5}/_{2} + {}^{1}/_{3}$.

Квантификатор ***очень*** может интерпретироваться с помощью операции концентрации, т.е. возведения в квадрат: **очень x** = $\int \mu^2_X(u)/u$.

Квантификатор ***не*** может интерпретироваться с помощью операции отрицания: **не x** = $\int (1-\mu_X(u))/u$ **.**

Тогда термин ***очень маленькое*** = $^{1}/_{1}+ {}^{0.16}/_{2}$, а ***не очень маленькое*** = $^{0.84}/_{2}+ {}^{1}/_{3}$.

Отношение для максиминного правила:

$$Rm'=(A \times B) \cup (\sim A \times C) = \int_{U\times V} (\mu_A(u) \wedge \mu_B(v)) \vee ((1-\mu_A(u)) \wedge \mu_C(v))/(u,v)$$

здесь A = ***маленькое,*** B = ***большое,*** C = ***маленькое.***

Пример вычисления значений элементов матрицы Rm':

$(u_1, v_1) = (1 \wedge 0) \vee (0 \wedge 1) = 0$, $(u_2, v_1) = (0.4 \wedge 0) \vee (0.6 \wedge 1) = 0.6$,

$(u_1, v_2) = (1 \wedge 0.5) \vee (0 \wedge 0.4) = 0.5$, $(u_2, v_2) = (0.4 \wedge 0.5) \vee (0.6 \wedge 0.4) = 0.4$,

$(u_1, v_3) = (1 \wedge 1) \vee (0 \wedge 0) = 1$, $(u_2, v_3) = (0.4 \wedge 1) \vee (0.6 \wedge 0) = 0.4$,

$(u_3, v_1) = (0 \wedge 0) \vee (1 \wedge 1) = 1$,

$(u_3, v_2) = (0 \wedge 0.5) \vee (1 \wedge 0.4) = 0.4$,

$(u_3, v_3) = (0 \wedge 1) \vee (1 \wedge 0) = 0$,

$$Rm' = \begin{vmatrix} 0 & 0.5 & 1 \\ 0.6 & 0.4 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix}$$

Тогда значение ***y'*** может быть определено следующим образом:

$$y' = \text{не очень маленькое} \bullet Rm' = \begin{vmatrix} 0 & 0.84 & 1 \end{vmatrix} \bullet \bullet \begin{vmatrix} 0 & 0.5 & 1 \\ 0.6 & 0.4 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0.4 & 0.4 \end{vmatrix},$$

т.е. ***y'*** = $^{1}/_{1}+ {}^{0.4}/_{2} + {}^{0.4}/_{3}$, что может быть интерпретировано (с некоторой натяжкой) как ***довольно-таки маленькое***.

Далее рассмотрим для указанных выше посылок арифметическое правило Ra':

$$Ra' = (\sim A \times V + U \times B) \cap (A \times V + U \times C) =$$

$$\int_{U\times V} 1 \wedge (1-\mu_A(u)+\mu_B(v)) \wedge ((\mu_A(u)+\mu_C(v))/(u,v) =$$

$$\begin{vmatrix} 0 & 0.5 & 1 \\ 0.6 & 0.8 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix}$$

Тогда значение ***y'*** при использовании арифметического правила может быть определено следующим образом

$$y' = \textit{не очень маленькое} \bullet \mathrm{Ra}' = \begin{vmatrix} 0 & 0.84 & 1 \end{vmatrix} \bullet$$

$$\bullet \begin{vmatrix} 0 & 0.5 & 1 \\ 0.6 & 0.8 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0.8 & 0.4 \end{vmatrix},$$

т.е. $y' = {}^{1}/_{1} + {}^{0.8}/_{2} + {}^{0.4}/_{3}$.

Вывод с использованием размытого бинарного правила приведен ниже:

$$\mathrm{Rb}' = (\sim A \times V \cup U \times B) \cap (A \times V \cup U \times C) =$$

$$\int_{U \times V} (1 - \mu_A(u) \vee \mu_B(v)) \wedge ((\mu_A(u) \vee \mu_C(v))/(u,v) =$$

$$\begin{vmatrix} 0 & 0.5 & 1 \\ 0.6 & 0.4 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix}.$$

Тогда значение y', может быть определено следующим образом

$$y' = \textit{не очень маленькое} \bullet \mathrm{Rb}' = \begin{vmatrix} 0 & 0.84 & 1 \end{vmatrix} \bullet \begin{vmatrix} 0 & 0.5 & 1 \\ 0.4 & 0.4 & 0.4 \\ 1 & 0.4 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0.4 & 0.4 \end{vmatrix},$$

т.е. $y' = {}^{1}/_{1} + {}^{0.4}/_{2} + {}^{0.4}/_{3}$, что может быть также интерпретировано как ***довольно таки маленькое***.

Последний нечеткий вывод проведем с использованием правила Танака-Мидзумото

$$\mathrm{Rgg}' = (A \times V \Rightarrow U \times B) \cap (\sim A \times V \Rightarrow U \times C) =$$

$$\int_{U \times V} (\mu_A(u) \to \mu_B(v)) \wedge ((1 - \mu_A(u)) \to \mu_C(v))/(u,v),$$

где $\mu_A(u) \to \mu_B(v) = \begin{cases} 1, \text{если } \mu_A \le \mu_B \\ \mu_B, \text{если } \mu_A > \mu_B \end{cases}$

$$\mathrm{Rgg}' = \begin{vmatrix} 0 & 0.5 & 1 \\ 0 & 0.4 & 0 \\ 1 & 0.4 & 0 \end{vmatrix}$$

$$y' = \textit{не очень маленькое} \bullet \mathrm{Rgg}' = \begin{vmatrix} 0 & 0.84 & 1 \end{vmatrix} \bullet \begin{vmatrix} 0 & 0.5 & 1 \\ 0 & 0.4 & 0 \\ 1 & 0.4 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0.4 & 0 \end{vmatrix},$$

т.е. $y' = {}^{1}/_{1} + {}^{0.4}/_{2} + {}^{0}/_{3}$, что интерпретируется как ***маленькое***.

2.2.2 Задания для контрольной работы № 2

Задание 1. База правил и рабочая память в продукционной системе имеет содержимое, заданное в вариантах. Проиллюстрировать графически механизм прямого и обратного логического вывода факта A. Обратите внимание на изменение содержимого рабочей памяти в процессе вывода. Проведите упорядочение правил вывода. Рассмотрите возможные конфликты при прямом и обратном выводе.

Вариант 1. База правил:
если F и D и E то B; если G то C; если B и C то A; если R то D; если S то A;
если F и G то M;
рабочая память: G, E, R, F.

Вариант 2. База правил:
если B и C и D то A; если E то B; если G и H то C; если F то B; если E то A;
рабочая память: G, H, D, F.

Вариант 3. База правил:
если B и C и D, то A; если F и G, то B; если H и D, то E; если E, то A;
рабочая память: G, H, D, F.

Вариант 4. База правил:
если C и D то B; если E то B; если F и G то E; если B то A; если H то C;
рабочая память: G, H, D, F.

Вариант 5. База правил:
если D то B; если F и H то D; если B и C то A; если G и R то E; если E то B; если F и G то C;
рабочая память: G, H, F.

Вариант 6. База правил:
если B то A; если E и F и D то B; если G то C; если H то C; если C то A; если D то H;
рабочая память: G, H, D.

Вариант 7. База правил:
если B то A; если E и F и G то C; если H то D; если C то B; если D то B; если E то D;
рабочая память: G, H, F.

Вариант 8. База правил:
если B и C то A; если E и D то A; если H то C; если R то D; если G то E; если F и G то B;

рабочая память: G, H, R.

Вариант 9. База правил:

если B то A; если F то B; если G то F; если C то A; если H то E; если D и E то B;

рабочая память: G, H, D.

Вариант 10. База правил:

если B и C, то A; если D, то B; если E, то B; если F то, C; если G то C;

рабочая память: E, F, G.

Задание 2. Постройте TLC-модель для определения понятия, заданного в варианте; поскольку слова, используемые в определении понятия, сами обозначают понятия, то, определив их, постройте некоторую структуру, определяющую каждое понятие через взаимосвязи с другими имеющимися понятиями. Рассмотрите не менее десяти понятий в сети.

Вариант 1. Понятие «студент».
Вариант 2. Понятие «профессор».
Вариант 3. Понятие «шкаф».
Вариант 4. Понятие «компьютер».
Вариант 5. Понятие «стол».
Вариант 6. Понятие «журнал».
Вариант 7. Понятие «книга».
Вариант 8. Понятие «ребенок».
Вариант 9. Понятие «трактор».
Вариант 10. Понятие «посуда».

Задание 3. Для всех десяти вариантов пусть имеется следующее правило:

если x – маленькое, то y – большое, иначе y – маленькое.

Вторая посылка и значения переменных ***маленькое, большое*** приведено в вариантах;

U = 1+2+3+4.

Найти значения ***y***, используя последовательно все четыре правила нечеткого вывода (максиминное, арифметическое, размытое бинарное и правило Танака-Мидзумото). Сравните результаты, насколько сильно они отличаются от ожидаемых.

Вариант 1. Вторая посылка: ***x – не большое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.8}/_{2}+ {}^{0.2}/_{3}$, ***большое*** = $^{0.2}/_{2}+ {}^{0.7}/_{3}+ {}^{1}/_{4}$.

Вариант 2. Вторая посылка: ***x – большое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.8}/_{2}+ {}^{0.2}/_{3}$, ***большое*** = $^{0.2}/_{2}+ {}^{0.75}/_{3}+ {}^{1}/_{4}$.

Вариант 3. Вторая посылка: ***x – очень большое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.7}/_{2}+ {}^{0.2}/_{3}$, ***большое*** = $^{0.25}/_{2}+ {}^{0.75}/_{3}+ {}^{1}/_{4}$.

Вариант 4. Вторая посылка: ***x – не маленькое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.8}/_{2}+ {}^{0.3}/_{3}$, ***большое*** = $^{0.2}/_{2}+ {}^{0.9}/_{3}+ {}^{1}/_{4}$.

Вариант 5. Вторая посылка: ***x – очень маленькое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.7}/_{2}+ {}^{0.1}/_{3}$, ***большое*** = $^{0.2}/_{2}+ {}^{0.8}/_{3}+ {}^{1}/_{4}$.

Вариант 6. Вторая посылка: ***x – маленькое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.75}/_{2}+ {}^{0.1}/_{3}$, ***большое*** = $^{0.25}/_{2}+ {}^{0.7}/_{3}+ {}^{1}/_{4}$.

Вариант 7. Вторая посылка: ***x – не маленькое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.75}/_{2}+ {}^{0.1}/_{3}$, ***большое*** = $^{0.25}/_{2}+ {}^{0.7}/_{3}+ {}^{1}/_{4}$.

Вариант 8. Вторая посылка: ***x – очень маленькое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.7}/_{2}+ {}^{0.1}/_{3}$, ***большое*** = $^{0.2}/_{2}+ {}^{0.8}/_{3}+ {}^{1}/_{4}$.

Вариант 9. Вторая посылка: ***x – не очень большое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.7}/_{2}+ {}^{0.2}/_{3}$, ***большое*** = $^{0.25}/_{2}+ {}^{0.75}/_{3}+ {}^{1}/_{4}$.

Вариант 10. Вторая посылка: ***x – очень большое.*** Значения переменных ***маленькое*** = $^{1}/_{1}+ {}^{0.7}/_{2}+ {}^{0.2}/_{3}$, ***большое*** = $^{0.2}/_{2}+ {}^{0.8}/_{3}+ {}^{1}/_{4}$.

2.3 Контрольная работа № 3

Цель работы – создание базы данных в среде ПРОЛОГ по выбранной предметной области.

Выполнение работы состоит из следующих этапов:

- выбор предметной области. Четкое (письменное) формулирование цели создания системы;
- формальное описание предметной области. В качестве формализма для описания должны быть выбраны таблицы;
- представление предметной области на языке ПРОЛОГ;
- формулировка запросов и оформление интерфейса.

Примерный круг предметных областей:

– расписание занятий;

– расписание движения транспорта (авто-, авиа-, железнодорожного);

– расписание приема врачами в поликлинике.

Для примера рассмотрим первую предметную область «Расписание занятий». В системе должны быть представлены отношения между четырьмя элементами: названием курса, временем, именем лектора и местом проведения занятий. Система должна выдавать сведения, на-

пример, увязывающие читаемый курс и фамилию лектора, день недели и фамилию лектора, занятость аудиторий по времени.

В качестве формализма для описания предметной области возьмем таблицу:

Ф. И. О. лектора	Предмет	Время			Место	
		День недели	Начало	Конец	Корпус	Аудитория
Ходашинский И.А.	ОИИиЭС	среда	9	11	ФЭТ	426
…	…	…	..	…	…	…

На языке ПРОЛОГ данный факт может быть представлен следующим образом:

represent("ОИИиЭС", time("среда",9,11),
lector("Ходашинский", "Илья", "Александрович"),
place("ФЭТ",426)).

Отношение, увязывающее читаемый курс и фамилию лектора, имеет следующий вид:

rel1(F,K):-represent(K,_,lector(F,_,_),_).

Отношение, увязывающее день недели и фамилию лектора, имеет следующий вид:

rel2(F,D):-represent(_, time(D,_,_),lector(F,_,_),_).

Отношение, увязывающее занятость аудитории во времени, имеет следующий вид:

rel3(A, K,D,H,K):- represent(_, время(D,H,K),_, place(K, A)).

ТРЕБОВАНИЯ К БАЗЕ ЗНАНИЙ:

- произведение количества полей на количество записей должно быть не менее 300 для табличного представления;
- число запросов к базе данных должно быть не менее пяти.

2.4 Контрольная работа № 4

В процессе выполнения работы предполагается создание экспертной системы. Выполнение работы состоит из следующих этапов:

- выбор предметной области. Четкое (письменное) формулирование цели создания системы;
- формальное описание предметной области. В качестве формализма для описания должны быть выбраны продукционные правила;
- представление предметной области на языке ПРОЛОГ;
- формулировка запросов и оформление интерфейса.

Примерный круг предметных областей:

1) лекарственные растения (грибы, ягоды);
2) покупка компьютера (автомобиля, квартиры);

3) кулинария;
4) породы домашних (диких) животных;
5) починка телевизора (автомобиля, компьютера);
6) лечение собаки (кошки, человека).

ТРЕБОВАНИЕ К БАЗЕ ЗНАНИЙ: количество правил должно быть не менее 30.

Рассмотрим семь животных распространенных пород. Ниже приведены продукционные правила, задающие описание животных. Здесь биологический класс – это птицы или млекопитающие.

Правило 1.
ЕСЛИ животное имеет волосы,
ТО это животное млекопитающее.

Правило 2.
ЕСЛИ животное дает молоко,
ТО это животное млекопитающее.

Правило 3.
ЕСЛИ животное имеет перья,
ТО это животное птица.

Правило 4.
ЕСЛИ животное умеет летать
И несет яйца,
ТО это животное птица.

Правило 5.
ЕСЛИ животное – млекопитающее
И ест мясо,
ТО это хищник.

Правило 6.
ЕСЛИ животное – млекопитающее
И имеет острые зубы,
И имеет когти,
И имеет посаженные впереди глаза,
ТО это хищник.

Правило 7.
ЕСЛИ животное – млекопитающее
И имеет копыта,
ТО это копытное.

Правило 8.
ЕСЛИ животное – млекопитающее
И жует жвачку,
ТО это копытное.

Правило 9.
ЕСЛИ животное – хищник
И имеет рыжевато-коричневую окраску,
И имеет темные пятна,
ТО это леопард.
Правило 10.
ЕСЛИ животное – хищник
И имеет рыжевато-коричневую окраску,
И имеет черные полосы,
ТО это тигр.
Правило 11.
ЕСЛИ животное – копытное
И имеет длинные ноги,
И имеет длинную шею,
И имеет рыжевато-коричневую окраску,
И имеет темные пятна,
ТО это жираф.
Правило 12.
ЕСЛИ животное – копытное
И имеет белый цвет,
И имеет черные полосы,
ТО это зебра.
Правило 13.
ЕСЛИ животное – птица
И не умеет летать
И имеет длинные ноги,
И имеет длинную шею,
И имеет бело-черную окраску,
ТО это страус.
Правило 14.
ЕСЛИ животное – птица
И не умеет летать,
И умеет плавать,
И имеет бело-черную окраску,
ТО это пингвин.
Правило 15.
ЕСЛИ животное – птица
И умеет очень хорошо летать, ТО это альбатрос.

Работа системы распознавания сводится к генерации гипотезы о принадлежности животного к тому или иному классу и к попытке подтвердить эту гипотезу. В нашем случае генерируется первая гипотеза: «распознаваемое животное – это млекопитающее». Для подтверждения

данной гипотезы необходимо, чтобы пользователь утвердительно ответил хотя бы на один из вопросов: «имеет ли животное волосы» или «дает ли животное молоко». Если положительный ответ получен уже на первый вопрос, то система генерирует следующую гипотезу «это млекопитающее – хищник». Если же положительный ответ не получен на первый вопрос, то система задает второй вопрос. Если на него получен положительный ответ, генерируется гипотеза «млекопитающее – хищник», если получен отрицательный ответ, генерируется гипотеза: «распознаваемое животное – птица». Процесс порождения гипотез и их проверки длится до тех пор, пока есть подходящие для этого правила. Описание таких правил приведено ниже:

```
rule(1,"животное","млекопитающее",[1]).
rule(2,"животное","млекопитающее",[2]).
rule(3,"животное","птица",[3]).
rule(4,"животное","птица",[4, 5]).
rule(5,"млекопитающее","хищник",[6]).
rule(6,"млекопитающее","хищник",[7, 8, 9]).
rule(7,"млекопитающее","копытное",[10]).
rule(8,"млекопитающее","копытное ",[11]).
rule(9,"хищник","леопард",[12, 13]).
rule(10,"хищник","тигр",[12, 14]).
rule(11,"копытное","жираф",[15, 16, 12, 13]).
rule(12,"копытное","зебра",[17, 14]).
rule(13,"птица","страус",[18, 15, 16, 19]).
rule(14,"птица","пингвин",[18, 20, 19]).
rule(15,"птица","альбатрос",[21]).
```

Первый аргумент в предикате ***rule*** – это номер правила, второй – род, третий – вид, четвертый – список вопросов, подтверждающий отношение род-вид.

Для того чтобы применить ту или иную продукцию, необходимо собрать факты, задав пользователю вопросы. Однако, прежде чем задавать вопрос, необходимо быть уверенным в том, что этот вопрос уже не был задан ранее при подтверждении других промежуточных гипотез. Информация о заданном вопросе и полученном на него ответе хранится в отношении ***fact(X, Y)*** динамической базы данных, где ***X*** – номер вопроса, ***Y*** – ответ на этот вопрос (**"да", "нет"**). Если вопрос был уже задан и на него получен положительный ответ, то вывод успешно продолжается, если же получен отрицательный ответ, то система сообщает о неуспехе. Множество задаваемых вопросов приведено ниже.

```
ask(X):- fact(X, "да"),!.
ask(X):- fact(X, "нет"),!,fail.
```

```
ask(1):- write("оно имеет волосы?"), !, complete(1).
ask(2):- write("оно дает молоко?"), !, complete(2).
ask(3):- write("оно имеет перья?"), !, complete(3).
ask(4):- write("оно умеет летать?"), !, complete(4).
ask(5):- write("оно несет яйца?"), !, complete(5).
ask(6):- write("оно ест мясо?"), !, complete(6).
ask(7):- write("оно имеет острые зубы?"), !, complete(7).
ask(8):- write("оно имеет когти?"), !, complete(8).
ask(9):- write("оно имеет посаженные впереди глаза?"), !,
         complete(9).
ask(10):- write("оно имеет копыта?"), !, complete(10).
ask(11):- write("оно жует жвачку?"), !, complete(11).
ask(12):- write("оно имеет рыжевато-коричневую окраску?"),
          !, complete(12).
ask(13):- write("оно имеет темные пятна?"), !, complete(13).
ask(14):- write("оно имеет черные полосы?"), !, complete(14).
ask(15):- write("оно имеет длинные ноги?"), !, complete(15).
ask(16):- write("оно имеет длинную шею?"), !, complete(16).
ask(17):- write("оно имеет белый цвет?"), !, complete(17).
ask(18):- write("оно не умеет летать?"), !, complete(18).
ask(19):- write("оно имеет бело-черную окраску?"), !, complete(19).
ask(20):- write("оно умеет плавать?"), !, complete(20).
ask(21):- write("оно умеет очень хорошо летать?"), !,
          complete(21).
```

Процедура ***recognition(X)*** занимает центральное место в программной реализации продукционной системы. Процедура состоит из трех предложений. В первом предложении генерируется гипотеза (***rule(N, X, Y, Z)***) и ищется ее подтверждение (***discover(Z)***); если гипотеза подтверждается, то выдается соответствующее сообщение, если выдвинутая гипотеза не подтверждается, то генерируется следующая. Второе предложение процедуры описывает ситуацию, когда пользователь на все вопросы ответил отрицательно и системе не удалось выдвинуть ни одной гипотезы. И наконец, третье предложение задает успешное окончание работы, когда была подтверждена хотя бы одна гипотеза.

```
recognition(X):- rule(N, X, Y, Z), discover(Z), !,
    write("_____", X, " -  ", Y, " по правилу ", N), nl,
    recognition(Y).
recognition("животное"):- write("это животное мне неизвестно"),!.
recognition(_).

discover([]).
```

```
discover([X|Y]):- ask(X), discover(Y).

complete(X):- nl, read(Y), assert(fact(X, Y)), Y="да".
```

Рассмотрим пример работы системы.

```
?- retractall(_), recognition("животное").
"оно имеет волосы?"
да
_____   животное – млекопитающее по правилу 1
"оно ест мясо?"
нет
"оно имеет острые зубы?"
да
"оно имеет когти?"
да
"оно имеет посаженные впереди глаза?"
да
_____   млекопитающее – хищник по правилу 6
"оно имеет рыжевато-коричневую окраску?"
да
"оно имеет темные пятна?"
нет
"оно имеет черные полосы?"
да
_____   хищник – тигр по правилу 10
```

Пример показывает полное распознавание животного.

Важный вопрос построения продукционной системы – это разработка структуры продукционного правила. Некоторые рекомендации приведены ниже:

- конструируйте правила, опираясь на структуру, присущую предметной области;
- используйте минимально достаточное количество условий при определении продукционного правила;
- избегайте противоречащих продукционных правил.

Каждое продукционное правило может быть независимым от других. Модульность правил позволяет легко модифицировать продукционную систему. Модификация заключается в добавлении, изменении, удалении правил и не затрагивает существующих процедур. Число правил в системе ограничено размерами памяти компьютера. Продукционные правила можно поместить и в динамическую базу данных, тогда возможно хранение правил и во внешней памяти компьютера.

ЛИТЕРАТУРА

1. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. – СПб.: Питер, 2000. – 384 с.

2. Корнеев В.В., Гарев А.Ф., Васютин С.В., Райх В.В. Базы данных. Интеллектуальная обработка информации. – М.: Нолидж, 2000. – 352 с.

3. Лорьер Ж.-Л. Системы искусственного интеллекта. – М.: Мир, 1991. – 568 с.

4. Мидзумото М. Нечеткое рассуждение с нечетким условным высказыванием вида «Если …то … иначе» // Нечеткие множества и теория возможностей. Последние достижения. – М.: Радио и связь, 1986. – С. 37–50.

5. Нечеткие множества в моделях управления и искусственного интеллекта. – М.: Наука, 1986. – 312 с.

6. Представление и использование знаний / Под ред. Х. Уэно, М. Исудзука. – М.: Мир, 1989. – 220 с.

7. Попов Э.В. Экспертные системы: Решение неформализованных задач в диалоге с ЭВМ. – М.: Наука, 1987. – 288 с.

8. Построение экспертных систем: Пер. с англ. / Под ред. Ф. Хейеса-Рота, Д. Уотермана, Д. Лената. – М.: Мир, 1987. – 441 с.

9. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. – М.: Мир, 1989. – 460 с.

10. Танака К. Итоги рассмотрения факторов неопределенности и неясности в инженерном искусстве // Нечеткие множества и теория возможностей. Последние достижения. – М.: Радио и связь, 1986. – С. 37–50.

11. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. – М.: Наука, 1983. – 360 с.

12. Ходашинский И.А. ПРОЛОГ в примерах и задачах. – Томск: Курсив, 2001. – 280 с.

СОДЕРЖАНИЕ