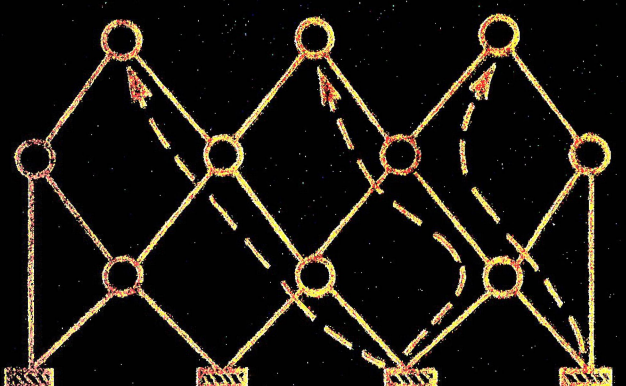
А.А. БАКАЕВ, В.И. ГРИЦЕНКО, Д.Н. КОЗЛОВ

# МЕТОДЫ ОРГАНИЗАЦИИ И ОБРАБОТКИ БАЗ ЗНАНИЙ



АКАДЕМИЯ НАУК УКРАИНЫ
ИНСТИТУТ КИБЕРНЕТИКИ им. В.М.ГЛУШКОВА

А.А. БАКАЕВ,
В.И. ГРИЦЕНКО, Д.Н. КОЗЛОВ

# МЕТОДЫ ОРГАНИЗАЦИИ И ОБРАБОТКИ БАЗ ЗНАНИЙ

КИЕВ НАУКОВА ДУМКА 1993

УДК 681.142.4

Анализируются методы проектирования и реализации интеллектуальных и прежде всего экспертных систем. Рассмотрены вопросы разработки и использования языка описания баз знаний, построения методов решения задач, способы работы с неточными, неполными, недостоверными и противоречивыми знаниями, методы объединения баз знаний и данных, а также подходы к реализации интеллектуальных систем.

Для разработчиков и пользователей систем, обладающих базами знаний, других специалистов в области искусственного интеллекта.

Аналізуються методи проектування та реалізації інтелектуальних і передусім експертних систем. Розглядаються питання розробки і використання мови опису баз знань, побудови методів розв'язання задач, способи роботи з неточними неповними, недостовірними та суперечливими знаннями, методи об'єднання баз знань і даних, а також підходи до реалізації інтелектуальних систем.

Для науковців та інженерів, що розробляють та використовують системи з базами знань, та інших фахівців у галузі штучного інтелекту.

**Ответственный редактор** *А.А.Морозов*

*Утверждено к печати ученым советом*
*Института кибернетики им. В.М.Глушкова АН Украины*

**Редакция физики и кибернетики**

**Редактор** *Б.В.ХИТРОВСКАЯ*

Б $\frac{1402070000\text{-}188}{221\text{-}93}$ 158-93

**ISBN 5-12-0003757-7**

# ПРЕДИСЛОВИЕ

Сегодня интеллектуальные системы олицетворяют одно из наиболее впечатляющих и приносящих огромную практическую отдачу приложений современных компьютерных технологий к решению многих задач. Эти системы представляют собой не только всевозможные компьютерные игры, начиная от детских "крестиков-ноликов" и заканчивая шахматными программами, обыгрывающими гроссмейстеров, но и с успехом справляются с такими задачами, как навигация космических кораблей, прогнозирование структуры и объемов сбыта для крупнейших корпораций мира, оценка политического и экономического состояния регионов планеты и многими др. Количество установленных и функционирующих систем этого типа в мире исчисляется десятками, если не сотнями, тысяч.

Существует множество синонимов понятия "интеллектуальная система", например, система, обладающая знаниями или базой знаний, система искусственного интеллекта. Данное понятие включает и экспертные системы, которые также построены на принципах искусственного интеллекта. Суть информационной технологии, воплощенной в интеллектуальной системе, можно выразить сравнением "работает как человек". Система не только имеет дело с теми же проблемами, с которыми люди сталкиваются в повседневной жизни или при выполнении своих профессиональных обязанностей, но и зачастую она внешне ведет себя как человек (задает вопросы, разъясняет ход рассуждений, выдает рекомендации) и во многом устроена так же, как и он. У нее есть "мозг", способный хранить и накапливать знания (база знаний), а также размышлять, используя эти знания при решении проблем (машина вывода). Она имеет "органы чувств" — средства взаимодействий с окружающим миром, себе подобными и людьми. Более того, многие компоненты интеллектуальной системы построены на принципах, в той или иной мере заимствованных у человека.

Однако ни один специалист, имеющий опыт работы с этими системами, никогда не скажет, что они способны моделировать поведение людей или хотя бы работу человека в некоторой узкой предметной области. Более точно все-таки говорить о способности имитировать деятельность человека, подражать ему. Дело в том, что отталкиваясь от способностей людей и пытаясь наделить ими компьютеры, исследователи в области искусственного интеллекта получили множество формальных и полуформальных методов, внешне соответствующих поведению человека, но основанных на других принципах хранения и обработки информации, свойственных не биологи-

ческим, а электронным системам. Эти методы внешне соответствуют поведению людей, все шаги внутри метода могут повторять действия человека, но детализация каждого шага при выполнении на компьютере неизбежно выглядит иначе, чем у человека. И если вспомнить мольеровского Журдена, которого удивляло, как сложно, оказывается, человек произносит звуки, то и в этом случае мы встречаем тот же подход, который применял учитель будущего дворянина. Используемый людьми алгоритм поведения также раскладывают на ряд шагов, каждому из которых соответствует аналог, допускающий эффективную реализацию на ЭВМ. Затем алгоритм выполняют на скоростной ЭВМ и в результате имитируют поведение человека, получая тот же результат, а иногда даже и более высокого качества, чем у среднего человека.

Сочетание различных методов, способных сымитировать поведение людей в ходе решения некоторой проблемы, образует технологическую цепочку обработки знаний в интеллектуальной системе. Не для всех подходов, применяемых людьми выявлены эффективные компьютерные аналоги. Однако в то же время многие известные методы искусственного интеллекта по качеству работы уже сегодня способны оставить позади (и оставляют) квалифицированных специалистов в той же предметной области, в которой работает компьютерная система. Эти методы, их сравнительные достоинства и недостатки, нерешенные проблемы рассмотрены в главах 1, 2.

В книге показано, что многие из методов и подходов, анализируемых в первых двух главах, можно объединить в единую технологическую цепочку представления и обработки знаний в различных интеллектуальных системах. Материал излагается на примере системы ЭКСПО — инструментального средства построения интеллектуальных систем, разработанного авторами в Институте кибернетики им. В.М.Глушкова АН Украины. Поскольку ЭКСПО служит инструментом для создания всевозможных баз знаний о различных предметных областях, то используемые в ней методы и подходы охватывают множество областей применения и многие типы задач. Спроектировать такую инструментальную систему равносильно созданию многих интеллектуальных систем, особенности которых должны быть учтены и для реализации которых должны быть предусмотрены разнообразные средства.

Кроме того, рассмотрены вопросы проектирования форм представления знаний (глава 3), методов логического вывода (глава 4) и способов работы с неопределенностью в системе ЭКСПО (глава 5). Попутно с методами представления и обработки знаний изложены особенности их применения для нужд той или иной предметной области. В главе 6 рассмотрены способы соединения баз знаний с базами данных, а глава 7 посвящена вопросам реализации на ЭВМ методов и подходов, изложенных в предыдущих главах.

Полагаем, что книга поможет пользователям и разработчикам интеллектуальных систем получить ответы на такие важнейшие вопросы: как

спроектировать и реализовать интеллектуальную компьютерную систему для решения выбранной задачи, какие средства искусственного интеллекта для этого использовать; как построить инструментальное средство для создания интеллектуальных систем и всевозможных баз знаний и, наконец, как настроить создаваемую интеллектуальную систему или инструментальное средство на нужды той или иной предметной области и решаемой задачи. За исключением главы 7 от читателя не потребуется каких-либо специальных знаний в области вычислительной техники и программирования для ЭВМ, однако для лучшего понимания терминологии, методов и особенно вопросов проектирования программного обеспечения, рекомендуется знакомство с основами искусственного интеллекта, экспертных систем и логического программирования в объеме, например, нашей предыдущей работы [БАК92].

*Авторы*

*ГЛАВА 1*

# ИНТЕЛЛЕКТУАЛЬНЫЕ КОМПЬЮТЕРНЫЕ СИСТЕМЫ

Интеллектуальные системы по своим возможностям далеки еще от способностей человека. В настоящее время перед ними стоит гораздо более скромная задача не моделирования, а имитации процессов принятия решений человеком. В отличие от людей компьютерные системы пока не в состоянии "рассуждать" обо всем, что их окружает, они проявляют свои способности лишь в ряде узких, строго очерченных предметных областей и решают лишь конкретные прикладные задачи. Каждая интеллектуальная система сегодня -- это компьютерный аналог специалиста узкого профиля.

Интеллектуальная система предназначена решать проблемы в выбранной предметной области на том же уровне компетентности, на каком работают специалисты в данной области, эксперты. Когда система реализована на ЭВМ и наделена знаниями экспертов любой специалист даже невысокой квалификации, пользуясь этой системой, сможет демонстрировать столь же высокие результаты работы, что и признанный эксперт. Главная цель создания системы в основном и состоит в том, чтобы поднять качество работы среднего специалиста или даже неспециалиста до уровня признанных авторитетов в своей области. Попутно возникает и ряд побочных эффектов, являющихся особенностью не столько интеллектуальных, сколько компьютерных систем вообще. Так, система, будучи реализованной на высокоскоростной ЭВМ, часто способна работать быстрее, чем рассуждает эксперт. Она не в состоянии что-либо упустить из виду, забыть, не обратить внимания на какой-то существенный факт, ее эффективность не зависит от погоды и не меняется по дням недели. Компьютерные интеллектуальные системы легко тиражировать, делая заложенные в них знания всеобщим достоянием, их можно быстро изменять, а также накапливать в них огромное количество информации на протяжении десятков лет. В результате совокупного влияния всех этих факторов система даже может оказаться более эффективным специалистом, чем человек.

В конце 80-х годов, спустя примерно 20 лет после начала исследовательских работ над интеллектуальными, и в частности экспертными, системами, один из ведущих специалистов Станфордского университета (США) и основоположников работ в этой области Фейгенбаум предпринял попытку оценить текущее состояние дел [FEI89]. Как выяснилось, наибольшее впечатление производит именно уровень эффективности этих систем, рассчитан-

ный как степень ускорения работы профессионалов и полупрофессионалов, использующих интеллектуальные системы. "Для удивления, казалось бы, нет оснований, ведь когда впервые появились компьютеры, никто не поражался, если ему говорили, что они умножают в тысячу раз быстрее людей, и следовательно, способности людей к арифметическим расчетам увеличились тысячекратно. Но даже таким специалистам, как я, никогда не приходило в голову, что способности людей рассуждать могут быть усилены на один или два порядка. Технологов учат рассматривать один порядок роста эффективности как магическую цифру, означающую революционные изменения... А мы везде обнаружили степень 10, увидели уровень 100 и даже один раз — 360. То есть, оценивая ускорение темпов работы людей, мы столкнулись с феноменальным ростом их производительности".

Не менее важно, что затраты на столь впечатляющий эффект несопоставимо малы, по сравнению с результатом. Например, в компании "Американ Экспресс" интеллектуальная экспертная система применяется для принятия решения о возможности выплаты наличных денег по предъявленной кредитной карточке. Все кредитные компании несут крупные финансовые потери из-за должников и подделки карточек, поэтому даже незначительное улучшение качества решений о расчетах приносит огромный эффект. "Американ Экспресс" оценивает прибыль, заработанную экспертной системой, в 27 млн дол. в год. Создание этой системы обошлось ей в 4 млн дол., и, следовательно, окупилось за два месяца первого года эксплуатации.

Подобные результаты получены и другими компаниями. Так, экспертные системы XCON и XSEL выбирающие конфигурацию ЭВМ и оформляющие платежные документы, принесли фирме "Диджител Эквипмент Корпорейшн" (ДЕК) (США) в 1987 г. 40 млн дол. чистого дохода, система LMS, установленная на радиоэлектронном заводе фирмы ИБМ в Бармингтоне (США), дает чистую прибыль примерно 100 млн дол. в год, а компания "Дюпон" оценила уровень рентабельности своих экспертных систем в 1500 %, тогда как в других областях она обычно считает удачей уровень в 35 % [FEI89]. В 1990 г. 600 интеллектуальных экспертных систем, установленных на предприятиях компании "Дюпон" по всему миру, принесли доход 75 млн дол., в 1991 г. ожидалось увеличение этой цифры минимум до 100 млн дол., а 50 работающих экспертных систем, включая XCON и XSEL, дают компании ДЕК 200 млн дол. прибыли ежегодно [MEA90]. При этом многие выгоды от применения интеллектуальных систем при расчете приведенных выше цифр вообще не учитывались. К примеру, APES — одна из систем фирмы ДЕК, помогающая инженерам при проектировании микросхем, подняла производительность проектировщика от уровня 200 логических элементов в неделю до 8000 в день. Уже первое применение этой системы позволило компании предъявить на рынок новую модель ЭВМ на шесть месяцев раньше расчетного срока, долгосрочные стратегические выгоды от чего трудно поддаются количественной оценке.

Среди многих факторов, определяющих столь большой эффект, главными все же являются богатые возможности технологии искусственного

интеллекта, воплощенной в каждой системе, способность этих систем аккумулировать знания людей и эффективно использовать их для решения сложных реальных проблем.

## 1.1. ЧТО ТАКОЕ ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА

Интеллектуальная система имеет дело с проблемами, решение которых человеком обычно связывают с наличием у него интеллекта. Она представляет собой компьютерный аналог специалиста, делающий в выбранной предметной области ту же работу, что и человек, которого в принципе она может заменить. Слово "интеллектуальная" в названии системы означает, что при ее построении использовались методы и подходы из арсенала средств искусственного интеллекта. Понятие интеллектуальной системы шире экспертной. Оно объединяет все экспертные системы, а также множество таких компьютерных систем, в которых технологии искусственного интеллекта применяются лишь частично, к примеру для реализации некоторых функций.

Интеллектуальная система обязательно имеет базу знаний, в которой в

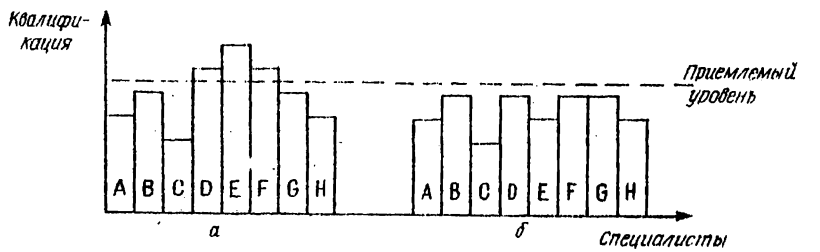



*Рис. 1.1.* Выбор экспертов в предметных областях:
*а* — область, в которой экспертами являются *D, E, F*; *б* — область, в которой эксперты отсутствуют

явном виде представлены знания людей. Используя эти знания, она осуществляет логический вывод при решении поставленных перед ней проблем. Как правило, это система, взаимодействующая со своим пользователем на понятном ему профессиональном языке, способная объяснить и обосновывать свои действия, имеющая средства накопления и изменения знаний.

Под специалистом, экспертом в выбранной предметной области понимают человека, обладающего необходимыми знаниями, навыками и опытом работы (рис. 1.1). На интеллектуальный уровень таких специалистов, на их знания и способы решения проблем ориентируют компьютерную систему. Знания человека запоминают в памяти ЭВМ, а его рассуждения имитируют с помощью специальной программы, в результате получают систему, работающую на уровне признанного специалиста в своей области. Менее квалифицированные люди, пользуясь системой, смогут приходить к тем же решениям проблем, что и эксперт, чьи знания воплощены в системе. А если в ней представить знания нескольких экспертов, то она сможет принимать реше-

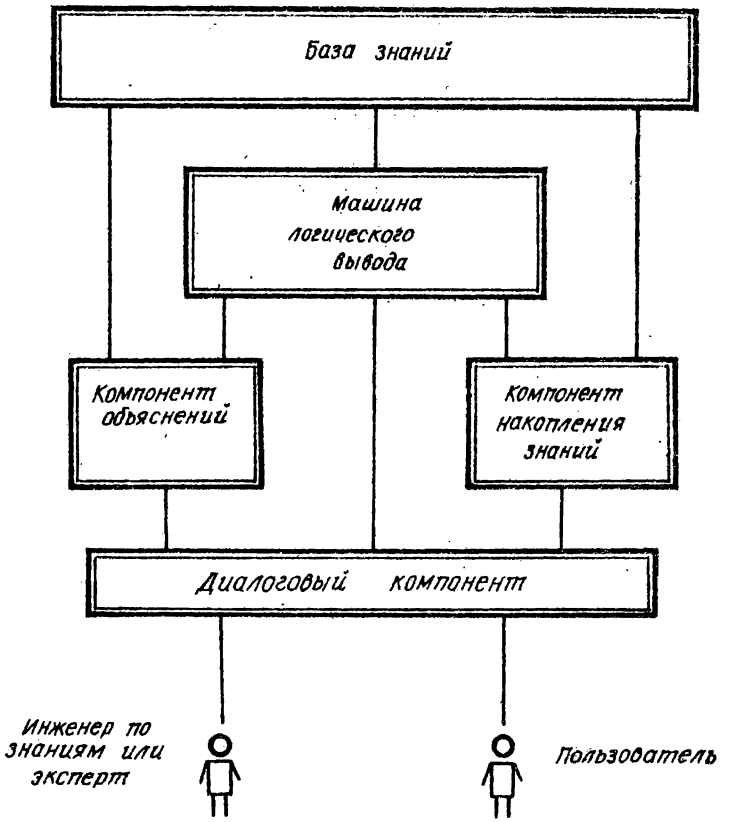



*Рис. 1.2.* Архитектура интеллектуальной системы

ния более квалифицированно, чем каждый из них отдельно. Так, созданная при участии Фейгенбаума экспертная система DENDRAL служит для определения структуры молекул химических соединений по данным спектрального и некоторых других видов анализа [ЭЛТ87, HAR85]. Система работает на интеллектуальном уровне примерно кандидата, доктора наук. Любой химик, общаясь с системой, способен столь же квалифицированно решать проблему, что и доктор наук, не пользующийся ЭВМ.

Разработанная в Массачусетском технологическом институте (США) система MACSYMA умеет выполнять символьное интегрирование и дифференцирование, решать всевозможные уравнения, системы уравнений и пр. Многие опытные математики признают, что проводят перед ее терминалами бо́льшую часть рабочего времени, столь захватывающими выглядят скорость и качество решения системой сложнейших задач [HAR85]. Знания системы получены от нескольких специалистов по различным разделам математики, что во многом и объясняет высокое качество работы, значительно превы-

шающее уровень среднего математика. Создатели системы расширяют ее до тех пор, пока она не охватит всех алгебраических проблем высшей математики, т.е. не будет явно превосходить способности любого одного эксперта в отдельности.

**База знаний.** Интеллектуальная система обязательно имеет обособленную базу знаний, в которой в той или иной форме сохраняются полученные от специалистов знания (рис. 1.2). В отличие от многих других типов компьютерных систем в данной знания представлены в явном виде — их можно распечатать, прочитать, изменить и снова записать в систему, не изменяя при этом ни одной программы. Знания, которыми наделяют систему, обычно складываются из фактов, характеризующих предметную область, и правил логического вывода, с помощью которых из одних фактов путем дедукции выводятся другие.

Например, рассмотрим одно из правил системы AI/RHEUM, диагностирующей болезни соединительных тканей в ревматологии [ANE82]:

| | | |
|---|---|---|
| Если | | у пациента бывают припадки |
| | или | психоз, |
| | или | органический мозговой синдром, |
| | или | кома, |
| то | | имеется серьезное заболевание центральной нервной системы. |

Когда в базе знаний обнаружится хотя бы один факт, встречающийся в условии правила, например, "у пациента бывают припадки", то сделанный в правиле вывод справедлив. Это правило на основании четырех фактов (по признакам припадков, психоза, указанного синдрома или комы) позволяет выдвинуть гипотезу о заболевании пациента. Гипотеза далее может считаться новым фактом, самостоятельно установленным системой, который после обработки правила помещается в базу знаний и позднее используется в ходе дедукции наравне с другими фактами.

Для представления знаний в интеллектуальных системах наиболее часто применяют правила продукций и фреймы [УЭН89, HAY85, FIK85]. Несколько реже встречаются семантические сети и различные вариации на "сетевую" тему — коннектионистские, нейронные сети [GAL85], сети логического вывода [QUI83], и другие методы, а также способы, основанные на логике предикатов [УЭН89]. Все эти и некоторые другие подходы служат для представления в базах знаний фактов и правил логического вывода

Семантическая сеть состоит из узлов, связанных между собой дугами. Каждый узел соответствует некоторому объекту или событию, а дуга — отношению между парой объектов или событий. С помощью такой сети, например, в системе PROSPECTOR [УЭН89, ЭЛТ87], помогающей геологам при оценке месторождений полезных ископаемых, моделируются отношения вида "биотит (1) — это слюда (2)", "слюда (2) — это силикат (5)", "силикат (5) — это минерал (6)" (рис. 1.3). Узлами сети в данном случае служат

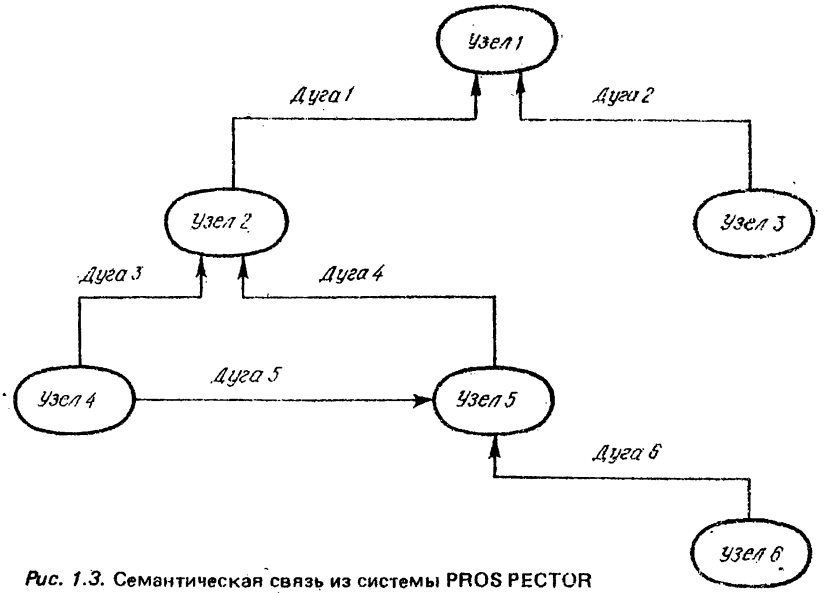



*Рис. 1.3.* Семантическая связь из системы PROS PECTOR

понятия "биотит" (1), "слюда" (2), "силикат" (5), "минерал" (6), а дугами между ними (1) — (6) — отношение "является" (связь типа is-a).

Другая разновидность сетей называется сетями логического вывода, или просто сетями вывода и также встречается в системе PROSPECTOR. В данном случае узлами служат логические заключения, например, благоприятный уровень эрозии (1), гипабиссальные условия образования (2), одновозрастные вулканические породы (3), а дуги (1) — (6) связывают с ними необходимые или достаточные условия. В такой сети каждый узел вместе с входящими в него дугами соответствует одному правилу логического вывода, а вся сеть задает совокупность правил базы знаний. Так, в приведенном на рисунке фрагменте сети встречается следующее правило из системы PROSPECTOR:

| | | |
|---|---|---|
| Если | | для района характерны гипабиссальные условия образования, |
| | или | в районе имеются одновозрастные вулканические породы, |
| то | | в районе имеется благоприятный уровень эрозии. |

Фреймы — это наиболее сложные структуры, сочетающие возможности правил логического вывода и семантических сетей. Обычно фрейм представляет собой описание одного объекта предметной области, например персонального компьютера для экспертной системы диагностики узлов ЭВМ (рис. 1.4). Относящаяся к этому объекту информация сохраняется в струк-

| Фрейм: ПЕРСОНАЛЬНЫЙ_КОМПЬЮТЕР | |
|---|---|
| Наименование: | IBM PS/2 |
| Изготовитель: | ИЗГОТОВИТЕЛЬ |
| Состояние: | неисправен,<br>если неисправен ПРОЦЕССОР<br>или неисправен МОНИТОР<br>или неисправна КЛАВИАТУРА<br>или неисправен ПРИНТЕР |
| Минимальная стоимость: | вычислить,<br>используя процедуру MIN |
| Максимальная стоимость: | вычислить,<br>используя процедуру MAX |

*Рис. 1.4.* Фрейм, описывающий персональный компьютер

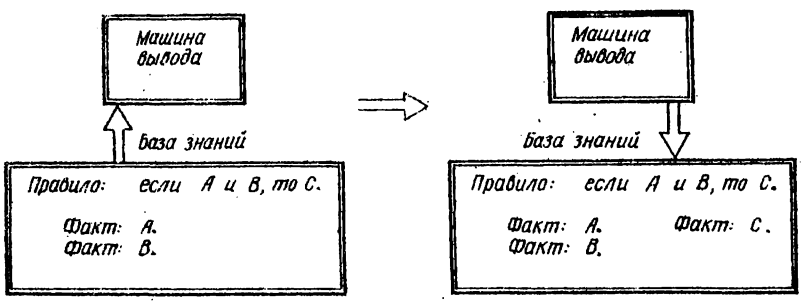



*Рис. 1.5.* Работа машины вывода

турных элементах фрейма, называемых слотами. Значением слота может служить константа (например, слот "наименование"), ссылка на другой фрейм (слот "изготовитель"), правило (слот "состояние") или процедура (минимальная и максимальная стоимости). Поскольку допускаются заполнители в виде правил логического вывода, а также ссылки фреймов друг на друга, то как форма представления знаний фреймы не уступают в отдельности ни правилам, ни семантическим сетям. Фактически они являются наиболее мощными структурами, используемыми в интеллектуальных системах, и наиболее часто применяются для создания больших баз знаний о различных предметных областях и решения сложных проблем.

**Машина вывода.** Решение проблемы возлагается на специальный компонент интеллектуальной системы, называемый машиной дедукции, машиной логического вывода или просто машиной вывода. Она берет факты, хранящиеся в базе знаний, и, используя правила, делает логические заключения

или инициирует какие-либо действия (рис. 1.5). Работа машины вывода направлена на решение проблемы, она имитирует ход рассуждений человека так же, как база знаний имитирует его память. При необходимости машина вывода выполняет не только логические, но и другие, например математические, операции, если они необходимы в ходе решения.

Общепринятые способы логического вывода, реализованные во многих интеллектуальных системах, заключаются в построении прямых и обратных цепочек рассуждений [БАК92, ЭЛТ87, HAR85]. Прямая цепочка начинается с анализа известных фактов и заключается в последовательном применении правил, сопровождающемся накоплением новых фактов, до тех пор, пока на ощупь не найдено решение проблемы (рис. 1.6). Обратная цепочка, напротив, означает перебор заранее известных решений для того, чтобы убедиться в том, какое из них соответствует имеющимся фактам. В логике предикатов и системах, основанных на ее приложении, эти методы называют соответственно поиском снизу вверх и поиском сверху вниз [КОВ90].

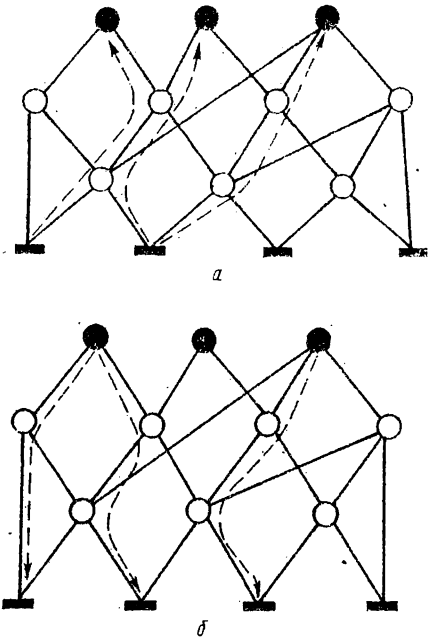


*Рис. 1.6.* Прямая (*а*) и обратная (*б*) цепочки рассуждений:
● — решения; ○ — правила; ■ — факты

Эксперты, на уровень работы которых ориентируют машину вывода, в большинстве случаев применяют сложные способы рассуждений, и для их повторения, как правило, недостаточно таких простых методов, как прямая или обратная цепочки. Большую помощь оказывает комбинирование нескольких различных методов в смешанную стратегию решения задачи, когда машина вывода, имитируя поведение эксперта, многократно переключается с одного режима работы на другой. Часто используют так называемые демоны, благодаря которым легко нарушается обычная последовательность рассуждений после выполнения условий "когда — известно", "когда — удалено", "когда — добавлено" и т.п. [УЭН89]. Во фреймовых системах нерегулярные способы вывода удобно задавать посредством механизма присоединенных процедур [FIK85, УЭН89], а с сетевыми структурами часто применяют специфические способы вывода, например, основанные на согласовании элементов сети [QUI83].

Многообразие методов логического вывода, способов решения проблем,

разработанных для интеллектуальных систем, объясняется той же причиной, что и использование многих форм представления информации в базах знаний, а именно сложностью задачи воссоздания на компьютере способов мышления людей, и в частности экспертов. Поставленная задача посильна, если выделять типы проблем и выявлять для них приемлемые методы решения, в целом соответствующие действиям экспертов. С прикладной точки зрения такой подход означает, что в каждом конкретном случае, создавая систему, необходимо выбирать из множества известных вариантов именно тот метод, который точнее отражает поведение эксперта. Разработка каждой новой формы представления знаний и нового способа решений проблем дает возможность охватить все больше предметных областей и больше типов задач.

**Организация диалога.** Отличительной чертой интеллектуальных систем является и их общительность. Большинство из них в ходе решения проблемы ведут интенсивный диалог со своим пользователем, обсуждая, уточняя и разъясняя поступающую в базу знаний информацию, возможные промежуточные решения и линии рассуждений, суть полученных результатов. Согласно имеющимся оценкам до 50 % объема программ в экспертных системах приходится на реализацию диалога [ВОПВ7].

Взаимодействие пользователя с системой, за редким исключением, ведется на подмножестве естественного языка, ограниченном нуждами конкретной предметной области. При этом широко применяются компьютерная графика, видеотехника — словом, все, что можно встретить в хорошей игровой программе, рассчитанной на работу с непрограммистом. Все делается для того, чтобы пользователю для общения с компьютерным экспертом не приходилось учить какой-либо промежуточный язык, например язык программирования, а достаточно было бы знания своего родного языка и терминологии в выбранной предметной области.

Однако кроме вполне понятного стремления к повышению культурного уровня, существует и более глубокая причина общительности интеллектуальных систем. Работа, которую им приходится выполнять, часто связана с огромным риском, например финансовым, если речь идет о банковской системе, или с опасностью для жизни людей в случае медицинской диагностики. Даже если система получает информацию от сенсоров и решает некоторые задачи без вмешательства человека, она обязана по первому его требованию объяснять и обосновывать свои действия, быстро выдавать всю известную ей информацию и подчиняться командам персонала. В сложных ситуациях она обязана согласовывать свои действия с персоналом, советоваться, предлагать на выбор несколько линий дальнейшего поведения, в этих условиях качество предоставляемой системой информации может сыграть ключевую роль для быстрого и качественного решения проблемы.

**Накопление знаний.** Эксперты в любой предметной области, как все люди, приобретают знания двумя способами: во-первых, читают книги, посещают лекции, общаются с коллегами, во-вторых, обучаются на собственном опыте. Интеллектуальные системы пока освоили только первый способ, при кото-

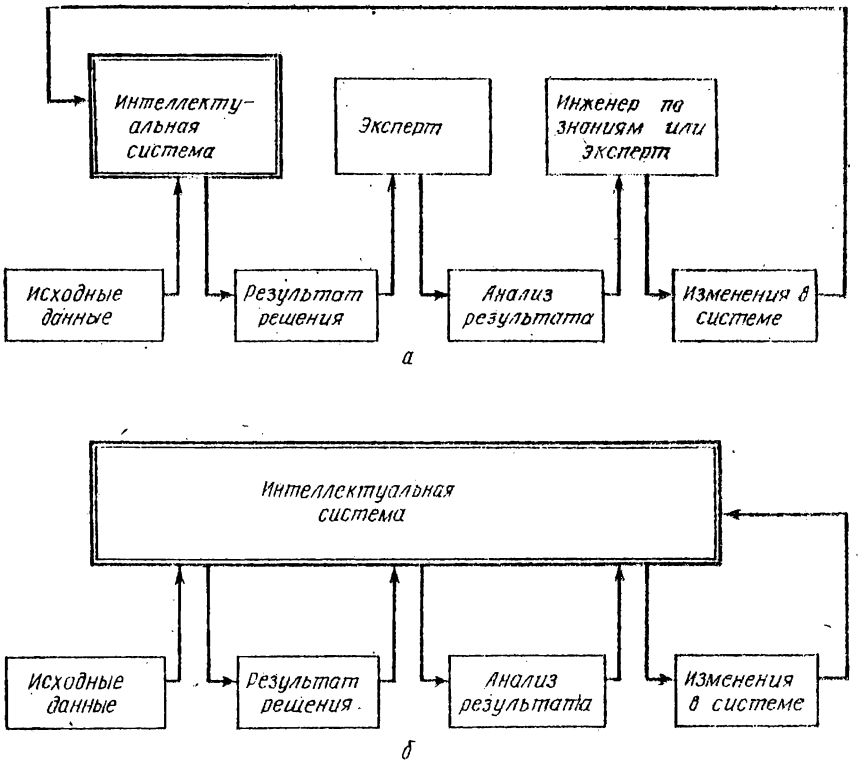



*Рис. 1.7.* Технологии накопления знаний:
*а* — традиционная; *б* — самообучения

ром знания поступают из окружающего мира от эксперта, либо от так называемого инженера по знаниям (рис. 1.7). Лишь единицы систем способны имитировать самообучение, да и то под контролем человека. Разработка действительно самообучающихся систем остается одной из наиболее серьезных задач, стоящих перед исследователями в области искусственного интеллекта [HAR85, FEI89]. Для решения этой задачи необходимо наделить компьютеры тем, что у людей называется здравым смыслом, а также большим количеством глубоких, причинно-следственных знаний.

Сегодня для наполнения, изменения и тестирования базы знаний необходим посредник, роль которого играет инженер по знаниям или эксперт. В функции инженера входит получение необходимых знаний от экспертов (для чего применяются специальные методики [УОТ89] или даже особые компьютерные системы [КАН85]), а также занесение информации в базу знаний. Кроме того, на него возлагается ответственность за рациональную организацию всей системы — ее базы знаний, структур управления логическим выводом, интерфейса пользователя и т.д. В сложной системе эти функ-

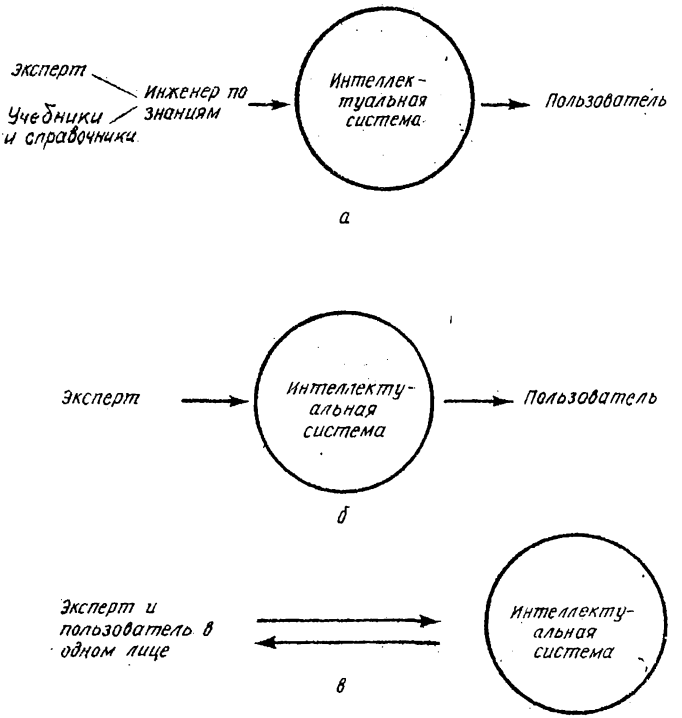



*Рис. 1.8.* Построение интеллектуальной системы:
*а* — инженером по знаниям; *б* — экспертом; *в* — пользователем

ции выполняет целый коллектив инженеров, куда входят проектировщики баз знаний и программисты.

Интеллектуальные системы, особенно созданные в последние годы, обладают столь развитыми средствами взаимодействия с окружающим миром, что функции инженеров по знаниям для них в состоянии выполнять непосредственно эксперты, от которых даже не требуется особой квалификации в области компьютеров и программирования. Результаты ряда обследований показывают, что в настоящее время примерно половина систем делаются непосредственно экспертами, так как все, что раньше должны были знать инженеры, системы теперь знают сами [FE189]. В данном случае эти системы сумели заменить экспертов по созданию самих себя (рис. 1.8). Однако о полном исключении инженеров речь пока не идет, так как без них по-прежнему не обойтись при построении очень сложных систем и тех, что стоят на передовом рубеже технологии искусственного интеллекта.

**Объяснения.** Качество работы системы, как и эксперта, во многом определяется способностью не только решать сложные проблемы, но и обосновывать свои действия. Более того, пользователю, на котором лежит ответственность за окончательное решение, непременно должна быть предоставлена

возможность убедиться в том, что система действительно решает поставленную проблему, делает это осмысленно и не совершает ошибок. Выполнением этих функций в интеллектуальной системе занимается компонент объяснений. Типичные действия, которые он выполняет по первому требованию пользователя, состоят в разъяснении того, какой информацией обладает система, как и на основании чего получено каждое логическое заключение (промежуточное или результат) и что система собирается делать с ним дальше. Попутно пользователь может запрашивать определения незнакомых терминов, обоснования корректности правил и другую информацию, необходимую для контроля за деятельностью системы. Чем полнее система объясняет свои рассуждения, тем больше уверенности у пользователя в правильности полученного решения. Кроме того, детальные разъяснения помогают отыскать и исправить логические ошибки в базе знаний и в способах рассуждений, а также облегчают понимание самой системы и сути решаемой проблемы. Объяснение своих действий настолько важно для интеллектуальных систем, что иногда даже считается обязательной их чертой [ALT85].

## 1.2. ВОЗМОЖНОСТИ КОМПЬЮТЕРНЫХ СИСТЕМ

Интеллектуальные системы, как правило, представляют собой большие программы для ЭВМ, хотя в последнее время благодаря развитию технологии производства интегральных схем отлаженные и проверенные временем системы все чаще реализуют в виде микропроцессоров. Независимо от принятого способа реализации компоненты системы выполняют функции, аналоги которых можно отыскать у человека. И все-таки возможности систем уступают способностям людей, особенно при решении сложных или необычных проблем хотя бы даже и в узкой предметной области. Главная причина сложившегося положения в том, что не для каждой возможности человека выявлен и опробован эффективный компьютерный аналог.

**Объем и структура знаний.** Человек по сравнению с интеллектуальной системой обладает гигантскими знаниями. Ограничения, существующие на допустимые размеры баз знаний систем, во многом определяют то, что компьютерные технологии применяются для решения недостаточно большого круга задач и не покрывают ряда областей, где успешно работают люди. Проблемы, связанные с размерами, возникают не вследствие трудности сохранения в памяти ЭВМ огромного количества данных — в этом компьютер не уступает человеку, — а в сложности эффективной обработки больших баз знаний: быстрой выборки необходимых правил (ассоциативного мышления), как можно более раннего сужения пространства решений (интуиции), быстрого пополнения и обновления знаний и т.п.

Существенно, что и структура знаний человека качественно отличается от организации имеющихся баз знаний (рис. 1.9). Во-первых, множество решений даже в узкопрофессиональных областях люди принимают на основании элементарного здравого смысла. Такие знания трудно формализовать и

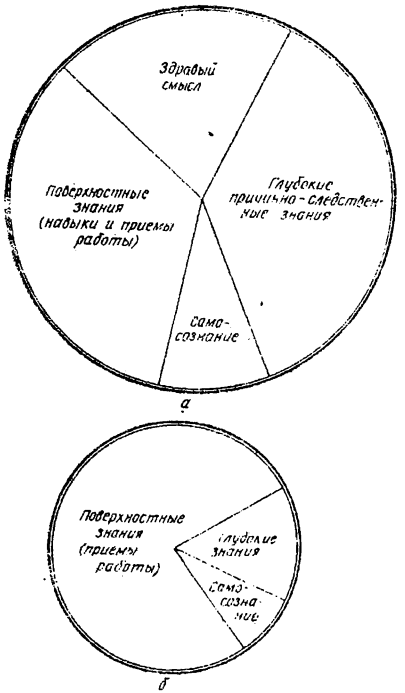



*Рис. 1.9.* Типичное соотношение структуры знаний эксперта (*а*) и интеллектуальной системы (*б*)

их приходится имитировать с помощью правил здравого смысла для каждого конкретного случая, т.е. увеличением размеров баз знаний. Во-вторых, эксперт обладает не только поверхностными, эвристическими знаниями о предметной области, но и глубокими причинно-следственными. Первые аккумулируют разнообразные приемы, основанные на опыте работы или полученные от наставников, из практических руководств, инструкций. Глубокие знания, приобретаемые в процессе теоретического обучения, включают знания о первопричинах и взаимосвязи явлений, а также отношениях между объектами. Поверхностные знания ускоряют решение несложных, часто встречающихся задач, тогда как глубокие позволяют найти выход и в ситуации, возникающей впервые. Интеллектуальным системам необходимы оба вида знаний, однако во многих случаях глубокие знания в них явно не представлены, а скрыты за формулировками правил логического вывода и процедур дедукции. Подобная имитация глубоких знаний с помощью поверхностных еще больше увеличивает размер базы знаний.

Кроме глубоких знаний о предметной области и решаемой проблеме системам необходимы сведения о самих себе, прежде всего о границах собственных познаний. В результате их недостатка существующие системы иногда могут приходить к неправильным решениям на периферии своих знаний. Эксперты достаточно легко обнаруживают подобную ситуацию и обращаются за помощью к специалисту высшей квалификации или специалисту смежной области. Компьютерные системы пока мало приспособлены к самоанализу, поскольку редко обладают знаниями о самих себе, хотя технология наделения систем такими знаниями известна и реализована, например, в обучающих системах (TEIRESIAS [ЭЛТ87] и др.).

На сложившемся уровне технологии акцент в основном делается на запоминании обширных поверхностных знаний, специфических для каждой предметной области. В этом смысле системы часто являются профессионалами большими, чем эксперты, но из-за недостатка здравого смысла и глубоких знаний все-таки хуже решают сложные проблемы. Недостаток здравого смысла и глубоких знаний сдерживает самообучение систем, приобретение ими знаний на основе собственного опыта. И хотя налицо ощутимые продви-

жения в исследованиях по обоим направлениям [LEN82, LEN90, FIN85], их результаты пока не получили широкого распространения в коммерческих разработках. Кроме того, без глубоких знаний не обойтись в тех системах, от которых требуются всестороннее объяснение и обоснование своих действий и которые должны уметь "проиграть" на модели предметной области несколько возможных линий рассуждений, чтобы обосновать выбранную линию рассуждений и отказ от всех остальных. В результате оказывается, что чем бóльшей долей глубоких знаний обладает система, тем выше качество решения проблем с ее помощью, тем больше ее потенциальные способности к самообучению и обоснованию принятых решений.

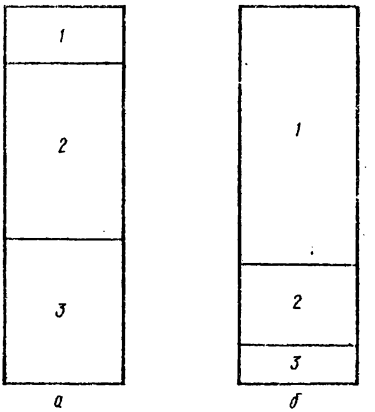



*Рис. 1.10.* Соотношение факторов, влияющих на время решения проблем экспертом (*а*) и интеллектуальной системой (*б*):

*1* — скорость обработки каждого правила; *2* — эффективная организация знаний; *3* — распараллеливание рассуждений

**Скорость работы.** Если систематичность, с которой машина вывода обследует возможные цепочки рассуждений, неизменно впечатляет экспертов, то скорость вывода логических заключений вовсе не обязательно превосходит темп рассуждений человека. Частично это объясняется недостатком здравого смысла, усложняющим принятие решений. Другая серьезная причина состоит в том, что экспертам свойственны методы рассуждений, более эффективные, чем прямые и обратные цепочки рассуждений. Более того, время решения проблем экспертом определяется не столько скоростью вывода отдельных заключений (выполнения каждого правила), которая, судя по всему, невелика, сколько применением рациональных способов запоминания знаний и эффективных стратегий управления выводом, а также распараллеливанием рассуждений (рис. 1.10).

Время обработки каждого правила машиной вывода определяется прежде всего быстродействием процессора ЭВМ, которое велико уже у существующих машин и продолжает возрастать буквально по часам. Обсуждаются предложения об измерении скорости процессоров в липсах — логических выводах в секунду, — для сокращения порядка цифр и точной оценки быстродействия с точки зрения новых задач. Большие перспективы откроет преодоление барьера параллельных вычислений, которое ожидается примерно тогда, когда будет разработана теория самообучающихся систем, т.е. к концу 90-х годов [HAR85]. Прогресс в области аппаратных средств позволит повысить эффективность многих традиционных методов, но, что более важно, в свете новых возможностей появятся и новые подходы к представлению и обработке знаний с помощью компьютеров. Магистральная линия движе-

ния в этом направлении заключается в приближении к способностям людей, однако не исключено, что они будут достигнуты на основе иных, не свойственных человеку, методов и подходов. Косвенным подтверждением этому может служить превосходство машин в области математических расчетов, хотя их процессоры работают не так, как вычисляют люди.

**Неточные и противоречивые знания.** При решении практически любой реальной проблемы исходные данные можно получить лишь с некоторой степенью точности и что даже на основе точных данных выводы можно сделать только с определенной степенью достоверности. В результате факты и правила логического вывода зачастую не имеют абсолютной надежности. Кроме того, обычной является ситуация, когда исходные данные и полученные с их помощью промежуточные результаты дедукции противоречат друг другу. Эксперты умеют работать в таких условиях, им удается либо получить несколько различных решений, надежно оценив достоверность каждого, либо даже отыскать единственное и правильное решение.

Интеллектуальные системы в целом освоили обработку неполных, неточных и недостоверных знаний, но хуже справляются с противоречиями. Для оценки неопределенности в них чаще всего используют либо различные неформальные измерители, либо вероятности. Например, в медицинской системе MYCIN, обеспечивающей диагностику менингита, с каждым фактом и правилом связывают значение так называемого коэффициента уверенности — число из интервала [−1, +1], в котором отрицательные значения неформально отражают уверенность в недостоверности, а положительные — в достоверности правила или факта [ЭЛТ87, HAR85]. Для дедукции с помощью таких измерителей применяют особые формулы, основанные на некоторых результатах теории вероятностей и теории множеств. В геологической системе PROSPECTOR, оценивающей месторождения полезных ископаемых, полученные от пользователя неформальные измерители пересчитывают в вероятности, которые затем обрабатывают по формуле Байеса, известной из теории вероятностей [ЭЛТ87, HAR85]. Каждый из этих методов использован не в одной интеллектуальной системе и давно доказал свою практическую полезность. Что касается критики обоих подходов, то часто упоминают их слабую теоретическую обоснованность, способную иногда поставить под сомнение точность дедукции [БАК92, ФОР87].

С этой точки зрения внимание разработчиков стали привлекать другие, более обоснованные методы, особенно базирующиеся на теории доказательств Демпстера — Шафера [SHA76] и теории возможностей Заде [ZAD78]. Отметим также достаточно совершенные подходы, использующиеся вместе с сетями логического вывода, в которых в ходе согласования сети естественным образом удается бороться с противоречиями [QUI193], улучшенный метод коэффициентов уверенности, базирующийся на теории Демпстера — Шафера [GOR85] и др. [УЭН89, PRA85, BUX89].

**Средства общения.** Многие интеллектуальные системы уже обладают удобными средствами общения как с пользователями, так и с экспертами или с инженерами по знаниям. Описания элементов баз знаний, задания сис-

теме, ее вопросы, ответы и объяснения обычно формулируются на естественном языке или на подмножестве естественного языка, используемом в предметной области. Так, инструментальное средство создания экспертных систем ROSIE [УОТ89] позволяет задавать правила логического вывода, формулируя их на естественном языке, а уже одна из самых ранних систем PROSPECTOR [УОТ89, ЭЛТ87, HAR85] обладала интеллектуальными средствами ведения диалога с пользователем. Однако не везде диалоговый компонент стал подлинно лингвистическим, чаще он полагается на такие традиционные средства, как меню, шаблоны и заготовки фрагментов фраз. Две наиболее известные интеллектуальные системы в этой области — HEARSAY III и INTELLECT — транслируют команды на естественном языке в запросы к базам данных, причем первая воспринимает звуковой ряд, а вторая — вводимый через дисплей текст запроса [HAR85]. Опыт их создания показал, что построение развитых лингвистических средств предполагает обработку знаний, т.е. применение тех же методов и подходов, которые используются при построении остальных компонентов интеллектуальных систем.

Появление у современных компьютеров развитых средств обработки графической информации — прежде всего дисплеев с очень высокой разрешающей способностью, графических рабочих станций, лазерных устройств ввода—вывода — предопределило относительное сокращение обработки лингвистической информации в пользу визуальной, зрительных образов. Во всех областях, использующих компьютеры, в конце 80-х годов произошел поворот от лингвистической информации к графической. Интеллектуальные системы с их изначальной ориентацией на непрограммирующих пользователей всегда находились в русле этой тенденции. Среди них особенно выделяются сложные фреймовые системы (такие, как ART, LOOPS или KEE [HAR85]), отображающие на экране не только структуру базы знаний и ход рассуждений, но и состояние объектов предметной области, схемы технологических процессов, шкалы измерительных приборов и пр. Системы DIPMETER ADVISOR [BOB87] и DRILLING ADVISOR [HAR85], помогающие персоналу при бурении глубоких скважин, визуально воспроизводят на экране дисплея конструкцию бура, послойную структуру скальных пород, показания всевозможных датчиков и результаты расчетов. Интеллектуальные системы в целом освоили выдачу такой визуальной онформации, а при создании для них средств чтения и понимания рисунков, графиков, текстовых документов используют тот же подход, что и при построении лингвистических средств, т.е. применяют методы искусственного интеллекта.

**Немонотонный вывод.** При монотонном логическом выводе все факты и промежуточные заключения справедливы вплоть до нахождения результата решения проблемы. Количество фактов и заключений, полученных машиной вывода и помещенных в базу знаний, монотонно возрастает по мере дедукции. При немонотонном выводе какие-то факты становятся недействительными к моменту решения проблемы и необходим частичный или даже полный возврат назад по цепочке рассуждений, сопровождающийся пересмотром уже выведенных заключений. Один из главных признаков немоно-

тонного вывода — это разрешение пользователю в любой момент приостанавливать работу системы и сообщать ей дополнительную информацию о проблеме. Немонотонность рассуждений важна в системах реального времени, а также в планирующих и таких, которые моделируют поведение сложных систем.

Для немонотонного вывода часто используют механизм так называемых активных значений [FIK85], реализованных с помощью демонов, в других случаях с каждым фактом связывают период его достоверности, срок службы [ВУЛ87]. Однако эффективность возвратов назад и последующего продвижения вперед по цепочкам рассуждений у многих систем пока уступает способностям экспертов. Главная причина этого состоит не в том, что машине вывода технически сложно повторить ход рассуждений и пересмотреть выведенные заключения, а в значительных затратах времени, превышающих время выполнения тех же функций экспертом. Для реализации немонотонности хорошо приспособлены семантические сети, элементы такого вывода встречались уже в системе PROSPECTOR, а также сети логического вывода, где механизм согласования равно учитывает монотонное, и немонотонное поступление фактов, не особенно отделяя их друг от друга.

**Краткосрочные цели.** Уже в настоящее время интеллектуальные системы, обладая обширными, хотя и недостаточно глубокими знаниями о предметных областях, конкурируют с экспертами по качеству и времени работы. Появление более совершенных компьютеров следующих поколений вместе с развитием технологий представления и обработки знаний может привести к тому, что машины сначала сравняются с экспертами, а потом и превзойдут их при решении многих сложнейших проблем, как превзошли людей при выполнении математических расчетов или в способности хранить и быстро обрабатывать гигантские базы данных.

Перечислим несколько краткосрочных целей или направлений развития интеллектуальных систем, продвижение в которых, как представляется, имеет первоочередное значение для совершенствования технологий, связанных с хранением и обработкой баз знаний: увеличение размеров баз знаний до десятков и сотен тысяч правил; повышение доли глубоких знаний о предметных областях и наделение систем знаниями о них самих, прежде всего о способах рассуждений и границах их собственных знаний; совершенствование вывода в условиях неопределенности и создание теоретически обоснованных методов количественной оценки достоверности принятых решений; разработка более сложных способов логического вывода, точнее моделирующих поведение экспертов, и повышение эффективности немонотонного вывода; увеличение скорости логического вывода как минимум на один-два порядка, а также разработка методов параллельных рассуждений; совершенствование средств общения лингвистических и ориентированных на обмен графической информацией.

Решение хотя бы некоторых из перечисленных проблем способно существенно расширить границы области применения компьютерных интеллектуальных систем. Следующий, качественно новый этап совершенствования технологии откроет преодоление барьеров распараллеливания работы ЭВМ и самообучения компьютерных систем.

*ГЛАВА 2*

# МЕТОДЫ СОЗДАНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Построение интеллектуальной системы, способной сравниться по результатам работы с человеком хотя бы в узкой предметной области, представляет собой длительный и трудоемкий процесс. На создание, например, системы MYCIN, специализирующейся на диагностике бактеремии и менингита, ушло более 50 человеко-лет, столько же — на разработку XCON, выбирающую необходимую покупателю конфигурацию компьютеров фирмы DEK. Трудоемкость построения системы MACSYMA, решающей алгебраические проблемы высшей математики, составила более 100 человеко-лет, а компания "Американ Экспресс" оценила стоимость одной своей интеллектуальной системы — в общем-то просто программы — в 4 млн дол.

Названные системы, кроме последней, создавались в крупных исследовательских центрах — соответственно в Станфордском университете, университете Карнеги-Меллона и в Массачусетском технологическом институте. Огромные затраты финансовых средств и времени квалифицированных специалистов во многом определились тем, что все эти системы являются пионерскими разработками, в ходе которых или параллельно с которыми создавались соответствующие инструментальные средства построения интеллектуальных систем. Так, из проекта MYCIN выросла первая, самая ранняя оболочка экспертной системы, названная EMYCIN и представляющая собой MYCIN с удаленной базой знаний (Empty MYCIN — пустая MYCIN); параллельно с XCON разрабатывались инструментальные средства, объединенные общим названием OPS и первоначально ориентированные на чисто исследовательские работы в области моделирования мышления людей. С помощью каждого из этих инструментальных средств позднее был реализован не один десяток интеллектуальных систем, построение которых обошлось примерно на порядок дешевле, чем "прародителей". Однако и первоначальные затраты на самом деле относительно невелики и не идут в сравнение с затратами времени на обучение новых экспертов или с трудоемкостью создания аналогичных программ, но не использующих принципы и подходы искусственного интеллекта. Несопоставимы они и с большим эффектом от применения этих систем.

## 2.1. СПОСОБЫ РАЗРАБОТКИ

При создании каждой интеллектуальной системы перед ее разработчиками стоит вполне определенная цель. Спектр целей наиболее широк для исследовательских и других некоммерческих приложений. На одной его стороне находятся системы, которые изначально создавались для моделирования мышления людей, например OPS [ HAR85], первоначально предназначенная для подтверждения гипотезы о том, что основную часть когнитивного поведения людей можно выразить посредством правил продукций, или AM [УЭН89], ориентированная на "открытия" в области арифметики целых чисел. На другой стороне сосредоточены узкоспециальные технические приложения, например LOOPS, которая сначала применялась как игровая программа для демонстрации графических возможностей компьютеров фирмы "Ксерокс" (США) (позднее она, как и OPS, переросла в мощный коммерческий продукт с тем же названием). Для исследовательских систем обычно характерны длительные сроки создания и неудачный результат ценится так же, как и положительный.

Коммерческие, промышленные разработки в отличие от исследовательских нацелены на быстрое создание эффективной компьютерной системы, работающей не хуже экспертов. Однако слово "быстрое" далеко не всегда означает короткий путь. Типичным примером может служить PUFF [HAR85] — несложная система, выполняющая первичную обработку данных медицинских анализов в пульмонологии. В ходе ее разработки широко применялся опыт MYCIN, а для программной реализации выбрана оболочка EMYCIN. После того, как система была создана и отлажена, из нее выбросили все лишнее, а оставшуюся часть реализовали на языке BASIC и соединили со стандартным медицинским оборудованием для диагностики болезней дыхательных путей. Система была создана "быстро", так как во многом копировала MYCIN и использовала мощные средства поддержки языка представления и обработки знаний EMYCIN, однако продвижение вперед выполнялось поэтапно и результат — интеллектуальное оборудование для диагностики — был получен далеко не назавтра.

**Выбор проблемы.** Разработка интеллектуальной системы обычно осуществляется в несколько этапов. Сначала рассматривается проблема, которую предстоит решить, и пытаются ограничить ее такими рамками, чтобы она, с одной стороны, представляла безусловную практическую ценность, а с другой — удовлетворяла определенным условиям, гарантирующим применяемость принципов искусственного интеллекта, а следовательно, и успешное завершение работ. Как правило, интеллектуальные системы наделяются знаниями об узких и сравнительно хорошо определенных проблемах. Ими обычно считаются те проблемы, информацию о которых можно изложить в виде книги. Другой критерий — все относящиеся к делу знания принципиально возможно пересказать по телефону, но если собеседник не сможет их понять, существует опасность, что позднее, через несколько лет работы над системой, не поймет и она.

Пример набора критериев [ЭЛТ87]:

предметная область должна быть узкоспециализированной, в которой знания не включают много здравого смысла;

проблема не должна быть ни слишком легкой для человека (скажем, требовать для решения нескольких минут), ни слишком сложной (требовать многих часов раздумий);

проблема должна быть ясно сформулирована, четко определены исходные данные и результат;

необходимо сотрудничество эксперта, который будет участвовать в создании системы.

Перечисленные критерии трудно выразить количественно, однако известны случаи, когда ошибки при выборе проблемы приводили к неудачному завершению проекта. По понятным причинам чаще сообщают о результатах иного рода. Так, разработчики XCON считают, что своим успехом они обязаны прежде всего тому, что система создавалась для компоновки ЭВМ VAX—11, а не PDP—11, имеющей больше внешних устройств [ЭЛТ87]. Однако после того, как система была создана, ее удалось расширить, включив знания о PDP—11. Более того, позднее была создана система XSEL, составляющая для XCON заявки на основании заказов покупателей [HAR85]. Таким образом, разработка сложной системы, включающей XCON и XSEL, выполнялась за несколько шагов, на каждом из которых предметная область и круг решаемых проблем расширялись.

Подобный подход облегчает построение сложных интеллектуальных систем, а также иллюстрирует еще одну важную цель их применения — использование системы как инструмента объединения и согласования знаний, полученных от многих специалистов или от специалистов в нескольких различных, не взаимосвязанных областях. В XCON, XSEL объединены знания инженеров по сборке ЭВМ и служащих, занятых контактами с покупателями и оформлением платежных документов. Этот же подход встречается и в системе PRIDE, занятой проектированием копировальных машин в компании "Ксерокс" [BOB87]. Здесь согласованы знания дизайнеров и инженеров по созданию механических компонентов машин, а эффект от применения системы частично определился резким сокращением переговоров между обеими группами специалистов.

**Реализация системы.** После того, как предметная область и проблема определены, выбирают способ реализации системы. При первом подходе ее можно запрограммировать "с нуля", использовав один или несколько языков программирования. В распоряжении разработчиков имеется богатый набор языковых средств, значительно отличающихся прежде всего своими возможностями, а также степенью сокращения в результате их использования затрат на программирование интеллектуальной системы. Перечень включает универсальные языки (Си, Паскаль и др.), а также специализированные языки искусственного интеллекта (Лисп или Пролог). На каждом из них можно реализовать, по видимому, любые необходимые способы представления знаний и методы логического вывода. Однако программирование

интеллектуальной системы на универсальном языке — задача намного более сложная и трудоемкая, чем создание ее же на языке искусственного интеллекта. В целом трудоемкость подобного подхода и требования к квалификации разработчиков достаточно велики, поэтому прямое программирование, как правило, применяют лишь при создании несложных, а также исследовательских, экспериментальных систем и больших промышленных разработок, но только там, где имеются программисты необходимой квалификации.

При втором подходе используют специальные инструментальные средства создания интеллектуальных систем, которые варьируют по уровню сложности и затрат на создание готовой системы. Сюда часто включают языки искусственного интеллекта, особенно Пролог и усиленные диалекты Лиспа. Далее следуют средства, называемые окружением, или средой (OPS-5, KEE, LOOPS, ART и др. [HAR85]), которые по сравнению с языками обеспечивают большее сокращение затрат. В них уже имеются программы, реализующие такие способы представления знаний, как правила или фреймы, а также программы, служащие строительными блоками при создании методов логического вывода. Программирование системы средствами окружения во многом напоминает возведение здания, которое строят не из кирпича, а из строительных конструкций. Еще более выгодным подходом к разработке является применение так называемых оболочек (1—stClass, Guru, EMYCIN, Микроэксперт и др. [FST87, GUR85, HAR85, ФОР87]), использование которых сводит затраты на программирование до минимума, а часто и до нуля. Это возможно благодаря тому, что основные проектные решения, касающиеся будущей системы, и прежде всего способы представления знаний и методы логического вывода, заранее встраиваются в программное обеспечение оболочки. В результате все программирование интеллектуальной системы сводится к наполнению базы знаний по принятым для этой оболочки правилам.

При третьем, наиболее простом подходе к разработке не используют никакого инструментального средства, а берут готовую систему, созданную и проверенную в других организациях. В этом случае все затраты на создание системы ограничиваются стоимостью ее покупки и, возможно, незначительными затратами на ее адаптацию. Если последние затраты велики, то это верный признак того, что следует использовать один из предыдущих подходов.

**Создание прототипа.** Когда речь идет о разработке, а не о закупке системы, после выбора подходящего способа реализации приступают к созданию прототипа. На этом этапе обычно реализуют небольшой, но достаточно детальный и типичный пример, отражающий специфику создаваемой системы. Прототип обязательно проверяют вместе с будущими пользователями, обращая особое внимание на их претензии к качеству диалога. После этого прототип зачастую переделывают заново.

Так, при создании прототипа для XCON его база знаний сначала содержала лишь 250 правил и охватывала 100 компонентов ЭВМ VAX-11. После

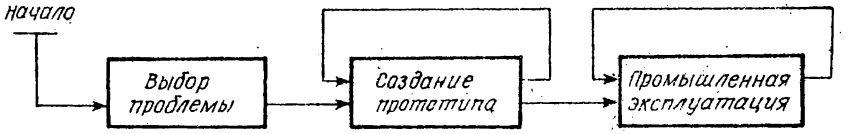



*Рис. 2.1.* Этапы создания интеллектуальной системы

испытаний количество правил было увеличено до 750 (420 компонентов). Затем прототип, реализованный с помощью инструментального средства OPS-4, был переделан под требования более совершенного средства OPS-5, а количество правил уменьшилось до 500 без ущерба для качества работы. Перед началом промышленной эксплуатации число правил было доведено до 850 (4000 компонентов), к 1986 г. – до 3500 правил, и база знаний системы расширяется до сих пор. После того как система сравнялась по качеству решений с экспертами, OPS-5 была переписана заново на языке Лисп для BLISS, с тем чтобы увеличить скорость работы по сравнению с первоначальным вариантом [BOB87, HAR85]. В результате окончательная версия XCON обеспечивает обслуживание одной заявки в среднем за 1,5 мин времени центрального процессора, а это такое быстродействие, которое трудно требовать от обычного эксперта.

**Эксплуатация.** Этап тестирования и переделывания прототипа иногда включает несколько итераций, в результате которых постепенно создается готовая к промышленной эксплуатации система (рис. 2.1). Однако и после начала работы не заканчивается развитие ее базы знаний, накопление информации параллельно с решением проблем. Считается, что процесс создания по-настоящему интеллектуальной системы имеет точку начала, но никогда не имеет точки конца. Подобно любому квалифицированному специалисту она должна непрерывно пополнять и улучшать свои знания на протяжении всего срока работы в роли эксперта.

## 2.2. РАЗНОВИДНОСТИ СИСТЕМ

Существует множество способов классификации интеллектуальных систем, каждый из которых делает акцент на какой-то одной области их применения или одной составляющей подхода, положенного в основу их построения. Прежде всего выделяют готовые к эксплуатации, собственно интеллектуальные системы и инструментальные средства создания готовых систем. Инструментальные средства в свою очередь подразделяют на специализированные языки программирования искусственного интеллекта, языки окружения или среды и оболочки. Готовые системы чаще всего классифицируют согласно области применения. Неполный список таких областей включает экономику и технику, медицину и биологию, геологию и космос, математику и военное дело [YOT89]. Однако с точки зрения конструирования интеллектуальных систем наибольший интерес вызывают не столько перечисление областей, где они успешно работают, сколько типы решаемых

проблем, методы представления знаний, размеры систем и способы их функционирования.

**Размеры баз знаний.** По размерам системы делят на три группы. В первую входят небольшие, узкоспециализированные системы, база знаний которых содержит примерно 50—500 правил. Обычно они реализованы на персональных ЭВМ или даже в виде микропроцессоров и решают несложные, но часто встречающиеся задачи. Многие фирмы и организации сознательно ограничиваются применением именно этих систем, иногда называемых интеллектуальными помощниками, внедряя их буквально на каждом рабочем месте, и за счет массовости получая огромную отдачу. Например, компания "Дюпон", преследующая такую политику, объявила, что средняя стоимость каждой из ее 600 систем, включая оплату экспертов за создание баз знаний, составляет 25 тыс. дол., а приносит каждая система, реализованная на персональном компьютере, в среднем 100 тыс. дол. в год [FEI89]. Для создания малых систем существует наибольшее количество оболочек (1—stCLASS, GURU, ESP Advisor, TIMM и др.) [FST87, GUR85, HAR85]), с чьей помощью даже эксперты, не знакомые с принципами программирования, могут быстро разработать для себя и других интеллектуального помощника.

Вторую группу составляют большие системы, чья база знаний содержит примерно до 5 тыс. правил и которые решают достаточно сложные проблемы в своей области. DENDRAL, XCON, MYCIN и PROSPECTOR — системы именно этого типа. Сюда же относится, например, INTERNIST (более поздняя версия системы называется CADUCEUS), которая позволяет диагностировать более 500 заболеваний внутренних органов человека [ЭЛТ87, HAR85].

Третью, наименьшую пока по численности группу составляют очень большие системы, у которых база знаний превышает 5—10 тыс. правил. Эти системы обычно программируются с помощью мощных инструментальных средств, таких, как KEE, LOOPS или ART [HAR85], и их разработкой заняты большие коллективы, включающие экспертов и инженеров по знаниям.

**Режимы работы.** По способу функционирования системы делят на работающие в режиме разделения времени, системы реального времени и обычные, установленные на обособленных ЭВМ и работающие в интерактивном режиме.

Согласно некоторым зарубежным оценкам наибольший рост наблюдается в группе систем реального времени и на них в ближайшие годы будет приходиться более трети рынка экспертных систем [ВУЛ87]. В основном эти системы заняты управлением технологическими процессами, наблюдением за состоянием пациентов в медицине или работой, связанной с мониторингом, в других сходных областях. Их появление стало возможным прежде всего благодаря увеличению скорости логического вывода заключений и совершенствованию техники немонотонного вывода. Эти системы чаще других реализуют в виде микропроцессоров, так как требования к скорости решения задачи максимально высокие.

В группу систем, работающих в режиме разделения времени, входят, к примеру, INTERNIST, MACSYMA, а также MOLGEN и выросшая из нее более поздняя система GENESIS (две последние заняты планированием экспериментов в молекулярной генетике [HAR85]). Все они установлены на компьютерах, имеющих выход в одну или несколько сетей ЭВМ. Пользователи получают доступ к соответствующим базам знаний, подключая свои терминалы к тем же сетям. Такой способ работы наиболее желателен для больших и очень больших систем. Во-первых, у каждого пользователя в отдельности не часто возникает необходимость решения достаточно узкой, но сложной проблемы. И, во-вторых, очень большую базу знаний легче поддерживать в актуальном состоянии, если она сосредоточена в одном месте, например в крупном исследовательском центре, изучающем именно эту проблему.

Третья, наиболее многочисленная группа объединяет все оставшиеся системы, а внутри нее в количественном отношении выделяются интеллектуальные системы для персональных компьютеров. Более того, согласно многочисленным оценкам внутри этой группы сектор разработок для персональных компьютеров устойчиво демонстрирует еще и опережающие темпы роста [MAR88]. Причем это объясняется не столько экстенсивными факторами (ростом количества компьютеров), сколько интенсивными (совершенствованием самих машин и технологий обработки знаний, позволяющих охватить все больше и больше новых задач). Некоторые зарубежные изготовители персональных компьютеров даже признают, что часть их наиболее производительных машин специально закупается для использования в интеллектуальных системах.

**Способы представления знаний.** В интеллектуальных системах в основном применяют три метода представления знаний — правила, семантические сети и фреймы. Соответственно и системы делят на работающие с правилами, сетевые, фреймовые и некоторые другие, где встречаются иные более специфические способы представления знаний.

Наибольшее распространение получили системы, основанные на применении правил логического вывода (MYCIN, DENDRAL, XCON, DRILLING ADVISOR и др.). Для их построения существует наибольшее количество инструментальных средств (EMYCIN, OPS-5, GURU и т.д.). Здесь знания представлены в виде конструкций:

Если < условие >, то < заключение >
или
Если < условие > , то < действие > .

Правила первого типа декларативны и предназначены для вывода логических заключений. Ниже приведен пример такого правила, взятый из системы MYCIN [SHO76]

| | | |
|---|---|---|
| Если | | окраска бактерий грамположительная |
| | и | морфология бактерий характерна для кокков |
| | и | форма колоний — цепочки, |
| то | | есть основания считать, что вид бактерий — стрептококк. |

С помощью декларативных правил системе сообщается логическое заключение, сделанное экспертом на основании перечисленных в условии правила исходных данных.

Правила другой разновидности более процедурны, они указывают машине вывода, какие действия ей следует предпринять, когда условие правила окажется истинным. Ниже приведено одно такое правило системы YES/MVS, разработанной фирмой ИБМ на основе инструментального средства OPS-5 для помощи операторам ЭВМ при управлении операционной системой MVS [УОТ89]

Если текущей задачей является поддержание достаточного объема свободной памяти для входной очереди заданий
и объем свободной памяти критически мал
и имеется линия связи с компьютером, которая активно принимает сообщения,
то послать команду отключить эту линию
и пометить режим приема сообщений по этой линии как "сообщения не принимаются".

Часто говорят, что посредством процедурных правил фактически программируется работа машины вывода, чтобы она шаг за шагом имитировала ход действий эксперта в процессе решения проблемы.

С точки зрения непрограммирующих пользователей и экспертов правила — наиболее удобный инструмент конструирования баз знаний, так как их легко формулировать, они просты и интуитивно понятны. Правила модульны, поэтому базу знаний легко изменять и расширять, удаляя из нее либо, напротив, добавляя туда правила по одному или группами. Объяснения, сделанные с помощью правил, понятны даже неподготовленному человеку. Существенно и то, что носителем всех этих достоинств служит физически одно и то же правило, которое составляется экспертом, читается и обрабатывается машиной вывода и выдается пользователю при объяснениях. Следовательно, каждое правило само по себе настолько информативно, что один и тот же его текст понятен как человеку (эксперту, пользователю или инженеру по знаниям), так и машине вывода.

В сетевых системах во главу угла ставится применение одной или нескольких разновидностей семантической сети. Так, в CASNET [УЭН89] с помощью многоуровневой сети моделируется развитие глаукомы и устанавливаются соответствия между симптомами, стадиями болезни, диагнозами, возможными методами лечения. В системах PROSPECTOR [УОТ89] и IDM [FIN85] сети также применяются для построения моделей предметных областей и задания правил логического вывода. В других системах иногда используется лишь какая-то одна разновидность сети, например сеть логического вывода, как в INFERNO [QUI83].

Базы знаний фреймовых систем, как правило, строятся на основе двух способов применения одних и тех же фреймовых структур. Одна группа фреймов служит для моделирования предметной области — объектов, поня-

тий и терминов, а также связей, отношений между ними. Другая хранит правила логического вывода, а связи между фреймами используют для задания структур управления выводом. В последнем случае в отличие от сетей узлами служат целиком правила рассуждений, а с помощью связей задают пути движения машины вывода внутри множества правил. При еще более сложном подходе в каждый фрейм помещают всю информацию о некотором объекте предметной области: описание этого объекта и все относящиеся к нему правила рассуждений. Такие структуры уже вплотную приближаются к тому смыслу фрейма, который вкладывал в это понятие Минский, впервые предложивший саму идею фреймов как средства описания всех знаний, относящихся к некоторой ситуации, объекту или зрительному образу. Фреймы обычно применяют в сложных системах, таких, как INTERNIST (CADUCEUS), MOLGEN, GENESIS [HAR85]. Для их построения предназначены инструментальные средства KEE, LOOPS, ART, KL—ONE и др. [HAR85].

Каждый из перечисленных способов представления знаний имеет и преимущества и недостатки. Правила особенно удобны для передачи поверхностных знаний о предметных областях, в частности в тех случаях, когда проблему можно решить за серию шагов, на каждом из которых известна и проверена опытом линия поведения. Однако выразительная сила правил недостаточна для определения терминов и понятий, а также описания объектов предметной области и связей между объектами. Главные недостатки правил лежат именно в области, эффективно покрытой сетями и фреймами. Фреймы являются наиболее мощными, но и самыми сложными структурами. В них удается объединить многие возможности семантических сетей и правил. Фреймы и сети больше подходят для представления глубоких причинно-следственных знаний, создания моделей предметных областей, представления иерархий концепций, понятий и объектов, последующие экземпляры которых наследуют свойства своих предшественников.

Названные структуры по своим возможностям не исключают, а скорее дополняют друг друга. Например, в системах, использующих правила, встречаются упрощенные аналоги фреймов для описания объектов предметной области, как это имеет место в OPS-5. В сетевых системах для работы с моделью предметной области обязательно присутствуют правила, иногда представленные в виде сети логического вывода (PPOSPECTOR). Фреймовые системы специально поддерживают средства описания объектов, обеспечивающие базовые дедуктивные возможности, облегчающие применение правил вывода, а сами правила разрешают задавать с помощью тех же самых фреймовых структур, подразумевая, что правило — такой же объект базы знаний, как и все остальные (KEE). Обычно система считается использующей правила, если не поддерживает ни сетевых, ни фреймовых структур. Систему, обладающую средствами работы и с правилами, и с сетями, для простоты называют сетевой, а имеющую и правила, и фреймы — фреймовой, тем самым подчеркивая наиболее сложный способ представления знаний из тех, что она в состоянии обеспечить.

Таблица 2.1. Примеры проблем

| Тип проблемы | Область применения |
|---|---|
| Классификация | Диагностика заболеваний в медицине по симптомам и выбор способа лечения<br>Диагностика технических систем и выдача рекомендаций по устранению поломок<br>Сравнение данных от сенсоров с ожидаемыми значениями и выбор способа реагирования<br>Диагностика и исправление действий обучаемого |
| Планирование | Планирование последовательности химических реакций для получения сложного органического соединения<br>Компоновка ЭВМ по заказу покупателя<br>Составление электрической или логической схемы по заданным входам и выходам |
| Проектирование и открытие | Разработка технического устройства с учетом требований и ограничений |

**Типы проблем.** Все проблемы независимо от того, в какой области специализируется интеллектуальная система, делятся на несколько типов. К наиболее известным относятся: классификация (диагностика и предписание); планирование, а также проектирование и открытие (табл. 2.1). Проблемы последнего типа — наиболее сложные, и в ходе их решения обычно сталкиваются с несколькими задачами планирования и классификации. Точно так же в процессе планирования почти всегда возникает необходимость проверить некоторый вариант плана на соответствие имеющимся ресурсам, выявить отклонения и выбрать лучший способ реагирования, т.е. решить задачу классификации.

Приведенные в таблице примеры для задачи классификации можно продолжить, поскольку к ним относится большинство интеллектуальных систем и именно для этого типа проблем существует наибольшее количество инструментальных средств. К задачам классификации обычно относятся все те, которые принципиально поддаются решению методом обратной цепочки рассуждений, т.е. путем перебора решений и проверки их на соответствие исходным данным. Однако по соображениям эффективности в них повсеместно используют прямые цепочки, как это делают эксперты. Типичным является подход, при котором рассуждение ведется в прямом направлении, после чего быстро сужают область возможных решений, а затем обследуют ее детально с помощью обратной цепочки. Именно этот прием позволил системе INTERNIST охватить диагностику более 500 заболеваний внутренних органов человека.

Сектор систем, решающих задачи планирования, несколько уже, но он устойчиво расширяется по мере совершенствования технологий обработки знаний. К нему можно отнести XCON, MOLGEN и GENESIS, а также два инструментальных средства: OPS-5, с чьей помощью создана XCON; и KEE, выросшую из проекта MOLGEN [HAR85]. Создание таких сложных систем сопряжено с большими трудностями, однако приносимый каждой из них эффект обычно сопоставим с результатом, полученным от применения многих классифицирующих разработок. Считается, что задачи этого типа принципиально нельзя решить с помощью обратной цепочки рассуждений, так как количество возможных вариантов пла-

на становится немыслимым даже при нескольких наименованиях ресурсов.

Проблемы проектирования, например новых изделий, и особенно открытия, переступают грань творчества, и применительно к ним разговор о каких-бы то ни было цепочках неуместен. Действительно проектирующие системы встречаются не часто. Один возможный подход к их построению воплощен в промышленной разработке, помогающей инженерам фирмы "Ксерокс" в проектировании механизма продвижения бумаги для копировальных автоматов [BOB87]. Однако это скорее советующая система, чем автоматический проектировщик, она не столько сама проектирует, сколько оценивает качество проектных решений инженера, подсказывает ему варианты решений и надежно оценивает преимущества и недостатки каждого. Тем не менее в процессе отладки системы, когда ее наполняли знаниями об уже выпускающихся машинах, она отыскала грубую ошибку в старом проекте и сумела безукоризненно ее исправить.

В специальной литературе зафиксировано первое открытие, сделанное интеллектуальной компьютерной системой, хотя она и не была предназначена для работы такого рода. Речь идет о системе DENDRAL, определяющей состав химического вещества по данным спектрального анализа. Система умеет оптимизировать свои знания, и по прошествии некоторого срока работы ее создатели заметили, что из имеющихся у нее знаний она вывела новое правило спектрального анализа, эффективное, но никогда не встречавшееся в литературе, а также успела проверить его, решая текущие задачи. Выявленная системой закономерность опубликована и получила признание специалистов-химиков.

Многие исследователи в области искусственного интеллекта считают, что будущие интеплектуальные системы на каком-то этапе своего развития смогут не только совершать открытия, но и имитировать такие сложные проявления человеческого сознания, как эмоции, если рассматривать их в качестве особого механизма управления поведением в условиях экстремальной ситуации [BOD89]. Пример такого необычного управления демонстрирует интеллектуальная система реального времени, когда она вплотную приблизилась к цели и должна решить, успеет ли завершить расчеты в отведенный промежуток времени или же ей стоит немедленно прекратить их и использовать другой, более грубый метод. Ответ на вопрос, действительно ли в этом случае система демонстрирует "страх", зависит от того, что считать эмоциями. Если рассматривать чувственную сторону дела, то, поскольку у компьютера нет сознания, бессмысленной становится сама постановка вопроса. Однако стоит только посчитать эмоциями специфические способы принятия решений, методы управления выводом в особых ситуациях, и становится ясно, что ими, как и другими методами, могут и будут обладать компьютерные системы.

## 2.3. ПРОГРАММНЫЕ И ТЕХНИЧЕСКИЕ СРЕДСТВА

Трудоемкость создания интеллектуальной системы зависит от того, насколько правильно выбран способ ее реализации и прежде всего тип инструментального средства, используемого для разработки программного обеспечения, а также другие программные и технические средства, окружающие систему или обеспечивающие ее работу.

**Языки программирования.** Программное обеспечение даже небольших интеллектуальных систем крайне редко разрабатывают на таких универсальных языках, как Си, Паскаль или Фортран. И дело не столько в большой трудоемкости создания программ, сколько в неприспособленности полученных программ к частым изменениям содержимого базы знаний и способов логического вывода. Для этой цели в основном используют языки символьного программирования Лисп и Пролог, обеспечивающие сокращение затрат на программирование интеллектуальных систем примерно на порядок по сравнению с универсальными языками.

Однако исторически сложилось так, что эффективность языков символьного программирования для традиционных ЭВМ, имеющих фон неймановскую архитектуру, часто оказывалась недостаточной, особенно при решении задач реального времени. Поэтому многие законченные и проверенные в эксплуатации системы позднее приходилось переписывать на одном из традиционных языков, компиляторы с которых "старше" и способны генерировать более эффективный машинный код. Такой подход повышал быстродействие программ, но в ущерб гибкости, способности систем изменяться с течением времени. Некоторые ранние системы даже эксплуатировались в двух экземплярах, один из которых был построен с помощью интеллектуальных инструментальных средств и легко поддавался изменениям, а другой — в виде трудноизменяемой традиционной программы. Все вносимые в систему корректировки сначала делались в первом экземпляре, тестировались, а лишь затем новая версия традиционной программы заменяла в эксплуатации предыдущую. По такой схеме была организована промышленная эксплуатация для XCON.

Позднее для традиционных ЭВМ были разработаны достаточно эффективные компиляторы с языков искусственного интеллекта (табл. 2.2), а также Лисп- и Пролог-машины, в которых конструкции соответствующих языков исполняются непосредственно аппаратурой. Эти машины способны обеспечить приемлемое время решения даже для задач реального времени. Благодаря их применению отпадает необходимость в переделывании программ, поскольку физически в одной и той же системе, написанной на языке символьного программирования, без труда удается совместить высокое быстродействие с необходимой гибкостью.

Несколько иначе обстоит дело с реализацией инструментальных средств создания интеллектуальных систем. Эти средства чаще, чем готовые системы, программируются на универсальных языках, но и здесь наблюдается тот же цикл — первая версия средства создается на языке искусственного

Таблица 2.2. Некоторые реализации языков искусственного интеллекта для персональных ЭВМ [MAR88]

| Оригинальное название | Перевод | Фирма-изготовитель (США) |
|---|---|---|
| Turbo PROLOG | Турбо ПРОЛОГ | "Борланд Интернешнл" |
| Prolog 2 | Пролог 2 | "Эксперт Система Интернешнл" |
| MProlog | МПролог | "Лоджиквэер Инк." |
| ExperProlog | ЭкспёрПролог | "Экспертеллидженс" |
| ExperCommon Lisp II | ЭкспёрКоммон Лисп II | " |
| Allegro Common Lisp | Аллегро Коммон Лисп | "Франц Инк." |
| Gold Common Lisp | Голд Коммон Лисп | "Голд Хилл Компьютерз Инк." |
| PowerLisp | ПауэрЛисп | "МикроПродактс Инк." |
| muLisp | маЛисп | "Софт Вэерхауз Инк." |

Таблица 2.3. Языки реализации некоторых инструментальных систем

| Система | Язык программирования и вид компьютера | Тип системы |
|---|---|---|
| 1—stClass | Паскаль, IBM PC | Простейшая оболочка экспертной системы [FST87] |
| EXPERT | Фортран, любая ЭВМ, где есть компилятор с этого языка | Оболочка экспертой системы [HAR85] |
| ES/P ADVISOR | Пролог, IBM PC | То же |
| G1 | Коммон Лисп, IBM PC/AT, TI Explorer, Sun 3, HP—9000 и др. | Инструментальный комплекс программ для решения интеллектуальных задач реального времени (окружение) [ВУЛ87] |
| KEE | Лисп, IBM PC/RT, Micro VAX II, VAX—750, Sun 3 и др. | Очень сложная инструментальная система (окружение) для построения баз знаний на больших, мини- и Лисп-машинах [FIK85] |

интеллекта, тестируется, отлаживается, проверяется в ходе ряда разработок готовых систем, а затем переписывается более эффективно. Появление и в этом случае новых компиляторов с символьных языков, генерирующих эффективный машинный код, а также Лисп- и Пролог-машин в целом позволило отказаться от переписывания программ. В результате совокупного влияния перечисленных выше факторов, по-видимому, для любого общеупотребительного языка программирования можно отыскать написанную на нем инструментальную систему (табл. 2.3).

**Типы ЭВМ.** Для реализации интеллектуальных систем используют практически все выпускаемые типы компьютеров, начиная от специализированных микропроцессоров и заканчивая большими универсальными ЭВМ. На выбор подходящего компьютера в каждом конкретном случае оказывают влияние факторы как финансового, так и технического характера и прежде всего предъявляемые к системе требования (необходимая скорость решения

задач, уровень компетентности, способ обмена информацией с окружающим миром и т.п.).

Типичным примером системы, реализованной в виде микропроцессора может служить безымянный интерпретатор результатов медицинских анализов, сделанных методом электрофореза [HAR85]. Первоначально прототип системы был реализован с помощью инструментального средства EXPERT и имел 10 правил логического вывода. Затем количество правил было доведено до 82, чтобы охватить 38 наиболее употребительных интерпретаций данных, т.е. причин отклонений результатов анализов от нормы. После того как на 256 наборах реальных данных была продемонстрирована 100-% приемлемость результатов интерпретации, систему реализовали в виде микропроцессора и встроили в стандартное электронное оборудование, которое с 1981 г. выпускается одной из фирм США – изготовителем аппаратуры для медицинской диагностики. Последующие изменения сначала вносят в версию, существующую в виде программы на языке системы EXPERT, а затем воплощают в новом микропроцессоре. Система не имеет даже собственного имени, как его нет и у любой другой микросхемы, входящей в состав электронного оборудования.

Разработчики систем для универсальных ЭВМ (больших, мини или персональных) останавливают свой выбор на том или ином типе компьютера в силу разного рода причин. Так, система YES/MVS, помогающая операторам управлять работой больших моделей ЭВМ, выпускаемых фирмой ИБМ, функционирует на том же самом компьютере, работой которого управляет. А, например, множество ранних разработок в области искусственного интеллекта, включая MYCIN и PROSPECTOR, делались на мини-машинах фирмы ДЕК по той простой причине, что эти недорогие ЭВМ были широко распространены в исследовательских центрах США и находились в неограниченном распоряжении разработчиков. Позднее многие ранние системы были переписаны для других типов ЭВМ и превратились в общеупотребительные коммерческие продукты, а сектор систем для мини-ЭВМ относительно сократился.

Однако наибольшее распространение получили интеллектуальные системы для персональных компьютеров. И если первоначально считалось, что они предназначены в основном для решения несложных, но часто встречающихся задач, то сегодня об этом даже не вспоминают. Широкий выбор готовых систем и инструментальных средств в сочетании с богатыми возможностями современных персональных компьютеров позволяют использовать их для решения сложных и даже очень сложных проблем. Об уровне сложности задач свидетельствует хотя бы тот факт, что стоимость готовой системы, предназначенной для выполнения на персональном компьютере, иногда достигает нескольких миллионов долларов, а максимальная цена только за инструментальное средство создания баз знаний на персональном компьютере, к примеру такого, как KEE, в зависимости от конфигурации доходит до 100 тыс. дол. за штуку.

Сложные системы для персональных компьютеров, как правило, реализуют на специализированных Лисп- или Пролог-машинах либо используют в

одной ЭВМ сочетание специализированного процессора с обычным. Эти машины дополнительно снабжают усиленными графическими процессорами и соответствующими видеотерминалами, лазерными сканерами и принтерами, плоттерами, видеокамерами и видеомагнитофонами. Стоимость таких специализированных компьютеров пока на порядок превышает цену на обычные ЭВМ. Однако с их помощью на персональном оборудовании удается объединить возможности интеллектуальных систем со средствами обработки больших объемов графической информации.

**Объединение баз знаний и данных.** Существует необходимость более тесного объединения баз знаний с базами данных, ставшими центральным информационным ресурсом многих предприятий и организаций. Их исторически сложившееся разделение, в результате которого средства обработки знаний применяются в основном для решения обособленных задач, является, по мнению многих специалистов, одним из основных факторов, сдерживающих распространение методов и средств искусственного интеллекта.

Объединение, с одной стороны, позволит наделить базы данных, в том числе существующие, интеллектуальными способностями и тем самым включить их в область действия новейших технологий обработки знаний. В то же время оно упростит построение сложных баз знаний и управление ими за счет использования сложившихся и достаточно развитых технологий обработки данных. На базы данных и системы управления базами данных могут быть переложены все функции, связанные с накоплением и хранением информации, организацией конкурентного доступа к ней, контролем непротиворечивости и целостности и т.п. Соединение с базами данных — это самый простой и естественный путь встраивания интеллектуальных средств в существующие технологии компьютерной обработки информации, многие из которых уже тесно интегрированы с базами данных. Решение проблемы объединения на практике будет означать, что преодолено последнее принципиальное препятствие для применения методов и средств искусственного интеллекта в разнообразных системах обработки данных. Более того, оно неизбежно приведет к появлению новых возможностей у систем обработки данных и у интеллектуальных систем. В частности, откроется очевидный путь построения распределенных баз знаний и их последующего объединения с помощью сетей ЭВМ в интеллектуальные системы национального масштаба. Такие системы будущего без труда смогут обладать гигантскими и всегда актуальными знаниями в широкой предметной области или даже в ряде областей, если связать в них все центры, где осуществляются приобретение и накопление знаний, их практическое применение и проверка.

Работы по достижению желаемого объединения ведутся в двух направлениях навстречу друг другу. Создатели средств управления данными совершенствуют свои системы, стараясь наделить их все большими дедуктивными способностями. Необходимости в отдельной базе знаний нет, считают они, "база данных должна все делать сама" [МОА87]. Так, одна из лучших дедуктивных систем управления базами данных POSTGRES, выросшая из известной реляционной системы INGRES, уже способна хранить многие тысячи

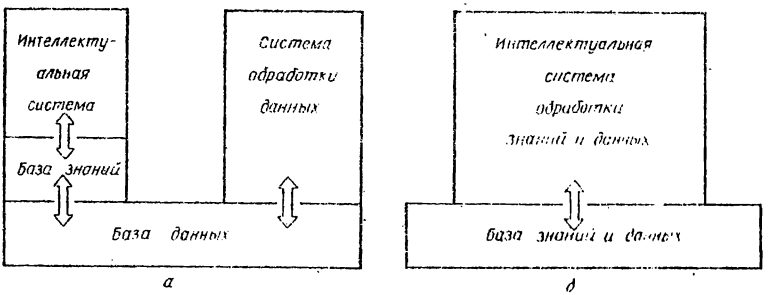



*Рис. 2.2.* Способы интеграции систем обработки знаний и данных: *а* — соединение через базу данных; *б* — полное объединение

правил логического вывода и осуществлять ограниченную дедукцию в прямом и в обратном направлениях [STO87]. Применение подобной системы существенно упрощает построение сложных баз знаний, однако даже она пока покрывает только малую часть возможностей систем искусственного интеллекта в части представления знаний и допустимых способов дедукции.

Двигаясь в противоположном направлении, исследователи в области искусственного интеллекта расширяют базы знаний своих систем, подключая к ним базы данных в основном как удобное средство хранения фактической информации. Ситуация, когда система управления базой данных применяется только для накопления и выборки фактов, а все дедуктивные возможности реализуются в надстроенной над ней системе искусственного интеллекта, является типичной для этого подхода (рис. 2.2). Так, в оболочке экспертной системы GURU любому факту в базе знаний разрешается поставить в соответствие заранее сформулированный запрос к реляционной базе данных, и как только машине логического вывода понадобится этот факт, он извлекается из базы данных с помощью заготовленного запроса [GUR85]. Такое применение базы данных позволяет эффективно хранить и обрабатывать огромное количество фактов. Однако база знаний складывается не только из фактов, но и из правил. Более того, факты настолько тесно связаны с правилами, что их искусственное разделение во многих существующих системах может рассматриваться как первая попытка действительного объединения баз знаний с базами данных.

*ГЛАВА 3*

# ФОРМЫ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Приступая к построению интеллектуальной системы прежде всего необходимо выбрать способ организации ее базы знаний. На выбор того или иного подхода оказывают влияние разные причины, главные из которых — это предъявляемые к системе требования и имеющиеся в наличии ресурсы технического характера, времени и финансовых средств. Эти требования и ограничения более или менее ясны, когда речь идет о создании некоторой вполне определенной промышленной разработки, полностью готовой к эксплуатации, например системы, составляющей конфигурацию конкретного компьютера вместо конкретного эксперта. Однако если иметь в виду создание некоторого инструментального средства, то в этом случае ситуация несколько усложняется, поскольку пользователи обычно считают, что оно должно быть максимально простым и универсальным и в то же время требовать минимальных затрат финансовых средств, памяти ЭВМ, времени и т.д. К сожалению, такого сочетания трудно достичь на практике, и обычно чем шире возможности системы, тем больших ресурсов она требует, и наоборот.

Ядро любой инструментальной системы составляет реализованный в ней метод организации базы знаний. Главное, что характеризует такой метод — это ответы на следующие три важнейших вопроса:

в какой форме представлены знания в системе. Например, имеют ли они вид правил или семантических сетей либо применена некоторая необычная форма представления или даже комбинация из нескольких форм. Ответ на этот вопрос помогает выяснить, какие собственно знания удастся занести в систему, удобно ли это будет сделать;

какие применяются способы обработки знаний, методы логического вывода. Ответ на этот вопрос дает представление о том, что система умеет делать с имеющимися у нее знаниями, с чем именно она справляется без особого труда, а что требует дополнительных усилий;

какой выбран метод работы с неопределенностью в знаниях, как обрабатываются неполные, неточные или противоречивые сведения. Иными словами, в состоянии ли система, работая с не до конца определенными знаниями, учесть многообразие окружающего мира и ориентироваться в нем не хуже, чем это делают люди.

Проектирование метода представления и обработки знаний для инструментальной системы представляет собой, по сути, выявление и согласование

ответов на эти три вопроса. Не следует думать, что процесс проектирования последовательно проходит три стадии: сначала выяснение форм представления знаний, затем создание способов обработки этих форм и, наконец, добавление методов работы с неопределенностью. Напротив, ответы на эти вопросы настолько тесно связаны между собой, что ход проектирования обычно напоминает ведение боевых действий во время войны, когда наступление на одном направлении неизбежно требует подтянуть вперед отставшие армии, затем какая-то одна из них снова прорывается вперед и уже за ней подтягиваются все остальные. В результате осуществляется постепенное продвижение вперед по всему фронту, т.е. создается метод, все элементы которого согласованы между собой, подогнаны друг к другу и поддерживают один другого.

В последующих главах рассматривается результат такого проектирования методов организации базы знаний для конкретной инструментальной системы ЭКСПО, а также некоторые вопросы реализации этих методов и подходов на ЭВМ. Настоящая глава содержит ответ на первый из трех названных выше вопросов: о формах представления знаний в системе, их особенностях и предоставляемых в этой части возможностях.

## 3.1. АРХИТЕКТУРА И ПРИНЦИПЫ ПОСТРОЕНИЯ ИНСТРУМЕНТАЛЬНОЙ СИСТЕМЫ

Одна из главных целей создания системы ЭКСПО состоит в изначальном согласовании противоречивых требований, которые обычно предъявляют к любому инструментальному средству — универсальность плюс максимум простоты при минимуме затрат ресурсов. Одновременное удовлетворение этих требований невозможно, и их согласование в ходе создания системы потребовало многих компромиссов, но все они основывались на том порядке целей, в каком те названы: прежде всего предпочтение отдавалось достаточной универсальности методов и подходов, затем легкости и простоте их применения с точки зрения пользователя, и, наконец, минимальным затратам времени и средств со стороны пользователей, а также ресурсов ЭВМ.

Ставилась задача получить внешне простой инструмент построения разнообразных баз знаний, обладающий тем не менее многими возможностями сложных систем. Для этого сложные средства описания и обработки знаний должны быть не только реализованы в инструментальной системе, но и представлены таким образом, чтобы ими мог воспользоваться даже непрограммирующий пользователь, например эксперт, создающий себе интеллектуального помощника и обладающий лишь начальными сведениями о компьютерах. Инструментальное средство должно быть настолько простым и понятным, чтобы с его помощью без особого труда и без написания каких-либо программ можно было бы создать не очень сложную интеллектуальную систему практически в любой предметной области. Иначе говоря, оно должно выглядеть, с одной стороны, примерно как универсальная оболочка эксперт-

ной системы — универсальная в том смысле, что не ориентирована на особенности какой-то одной предметной области или группы задач. В то же время инструментальное средство, сохраняя универсальность, должно обеспечить эффективное решение достаточно сложных проблем. Для достижения этой цели квалифицированным инженерам по знаниям и программистам разрешается расширять его возможности, увеличивая перечень функций, которые оно в состоянии исполнять. В этом случае инструментальное средство выступает уже не как оболочка, а как окружение или среда, в которых программируются новые возможности, например специфические методы дедукции, эффективные для определенного круга задач. Однако после того как новый метод присоединится к уже имеющимся, система должна допускать его применение столь же простыми средствами, как это имеет место для всех остальных ее методов. Расширение набора методов системы происходит так, как это принято в окружении или среде, т.е. путем программирования, но применение расширенного набора методов осуществляется средствами высокого уровня, что обычно свойственно оболочкам. Такая инструментальная система допускает неограниченное расширение своих функций и может быть использована для создания не только экспертных систем, но и многих других систем, где возникает необходимость хранения и обработки знаний. Иными словами, инструментальное средство в одних случаях можно использовать как оболочку, в других — как окружение, в котором программируются интеллектуальные системы, и применяться для создания замкнутых, обособленных систем, а также компонентов, ответственных за организацию и обработку баз знаний в некоторых более крупных системах.

При этом инструментальное средство должно обеспечить достаточную универсальность способов представления и обработки знаний, чтобы охватить множество предметных областей и способов применения. Кроме того, в разумных пределах следует обеспечить минимизацию возможных ресурсов пользователей прежде всего легкость освоения и применения системы, а также небольшую потребность в ресурсах компьютера — внешней и внутренней памяти, времени решения задач.

**Программные и технические средства.** Начиная создание системы, отвечающей перечисленным выше требованиям, необходимо было принять ряд решений, касающихся программных и технических средств, с помощью которых разрабатываются и позднее будет работать система. Прежде всего выбор был остановлен на языке программирования Пролог. Во-первых, для многих типов ЭВМ, начиная от персональных и заканчивая большими универсальными компьютерами, существуют эффективные компиляторы с этого языка, мало в чем уступающие компиляторам с других языков высокого уровня. Большое разнообразие типов машин, где работают компиляторы с Пролога, обеспечит, по нашему мнению, машинную независимость системы и возможность ее применения практически на любом общеупотребительном компьютере. Во-вторых, уже появились и все больше распространяются Пролог-машины, в которых конструкции этого языка исполняются непосредственно аппаратурой. Наша система после небольшого изменения

синтаксиса программ может быть использована и на этих компьютерах, и тогда эффективность решения ею задач возрастет многократно. Кроме того, в многочисленных проектах ЭВМ пятого поколения, реализуемых в нашей стране и за рубежом, именно Пролог выбран базовым языком создаваемых машин, и наша ориентация на программные средства пятого поколения позволяет учитывать тенденцию будущего, подготовиться к появлению качественно новых машин. В-третьих, имея некоторый опыт программирования на Прологе, мы были согласны с оценкой, что применение этого языка снижает трудоемкость кодирования и отладки интеллектуальных программ примерно на порядок. Однако у нас не было опыта создания крупных систем на Прологе, и мы хотели убедиться, что та же оценка сохранится при разработке многих разнообразных программ и их объединении в большую систему. Наши ожидания оправдались — примерно десятикратное сокращение объемов программирования и соответственно времени, проводимого перед дисплеями ЭВМ, мы ощутили на себе.

На персональных компьютерах типа IBM PC/AT было решено реализовать эталон системы, а варианты для других типов машин получать путем изменения синтаксиса программ под требования соответствующего компилятора. Работа, связанная с изменением программ на Прологе, не так сложна, как может показаться на первый взгляд, поскольку синтаксис языка чрезвычайно прост, программы отличает лапидарный стиль и они примерно на порядок короче программ, написанных на других языках высокого уровня и делающих ту же работу. В этих условиях изменение синтаксиса можно поручить другим программам, которые также закодировать на языке Пролог, первоначально появившемся именно как удобное средство трансляции с одного естественного или искусственного языка на другой.

**Организация базы знаний.** На следующем шаге было решено выделить два уровня представления знаний в системе — внешний и внутренний. Внешний уровень предназначен для взаимодействия с людьми, работающими с системой: пользователями, инженерами по знаниям, экспертами и т.д. Здесь знания имеют форму, понятную и удобную людям. В частности, это предложения, составленные из ограниченного набора слов русского языка и знаков препинания. Внешний уровень ориентирован на эффективное взаимодействие с человеком. В отличие от него внутренний уровень предназначен для эффективного решения задач компьютером и ориентирован на нужды ЭВМ. Понятные человеку предложения представлены в таком виде, который сокращает затраты памяти ЭВМ и ускоряет обработку информации программами на языке Пролог. Перевод описаний знаний из внешней формы во внутреннюю выполняет компилятор базы знаний, входящий в состав комплекса программ системы ЭКСПО. Компилятор как бы скрывает от пользователей тот факт, что в системе имеются еще какие-то структуры, кроме внешних, построенных из слов русского языка. И поскольку пользователь нигде не взаимодействует с внутренними структурами базы знаний, их рассмотрение отложим до главы 7, где рассмотрены эта и другие детали реализации системы. Вплоть до главы 7 речь идет лишь о внешних формах описания знаний,

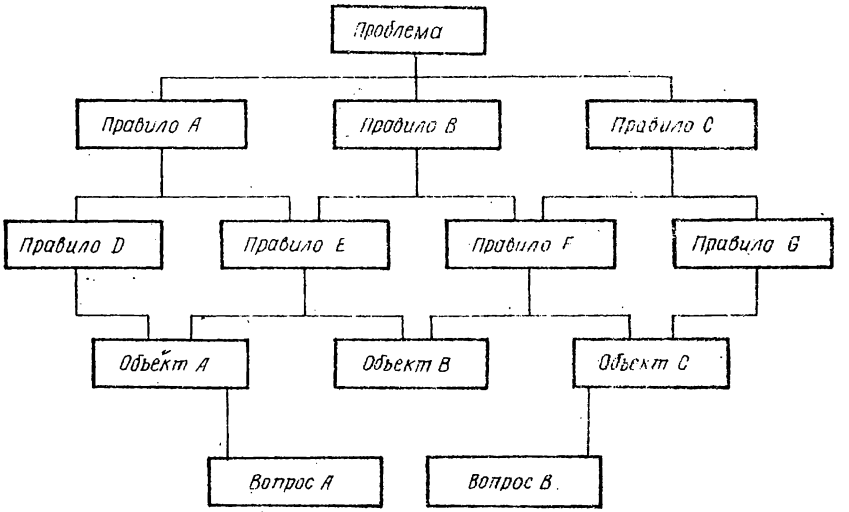



*Рис. 3.1.* Структура базы знаний системы ЭКСПО

т.е. о тех структурах, с которыми приходится иметь дело пользователям, инженерам по знаниям и другим людям, работающим с инструментальной системой.

На внешнем уровне для представления знаний используется несколько типов описателей, в которые помещается информация об объектах предметной области, о правилах логического вывода, способах решения проблем и т.п. (рис. 3.1). Прежде всего база знаний содержит описатели объектов предметной области, внешне напоминающие обычные фреймы. Каждый описатель включает информацию об одном объекте, свойства которого описываются в структурных элементах описателя, подобных слотам. Для рассуждений об этих объектах в базу знаний помещаются правила логического вывода. С их помощью логически выводятся новые факты, например на основе анализа некоторых известных свойств одних объектов определяются ранее неизвестные свойства других. Описатели вопросов обслуживают получение информации извне системы. Так, если некоторое свойство объекта заранее неизвестно и его следует выяснить у пользователя, то в описатель помещают текст вопроса, перечень допустимых ответов, подсказки, разъяснения вопроса и другие сведения. Обобщенный алгоритм решения задачи, с которой системе предстоит иметь дело, содержится в описателе проблемы. Здесь указывают, что требуется найти, как искать и что делать с результатом дальше. Кроме того, имеются и некоторые другие описатели знаний, которые рассмотрены ниже.

Многие возможности описателей объектов и правил соответствуют тем, которые свойственны обычным фреймам и правилам, однако описатели в отличие от реализаций обычных структур в других системах не предназначе-

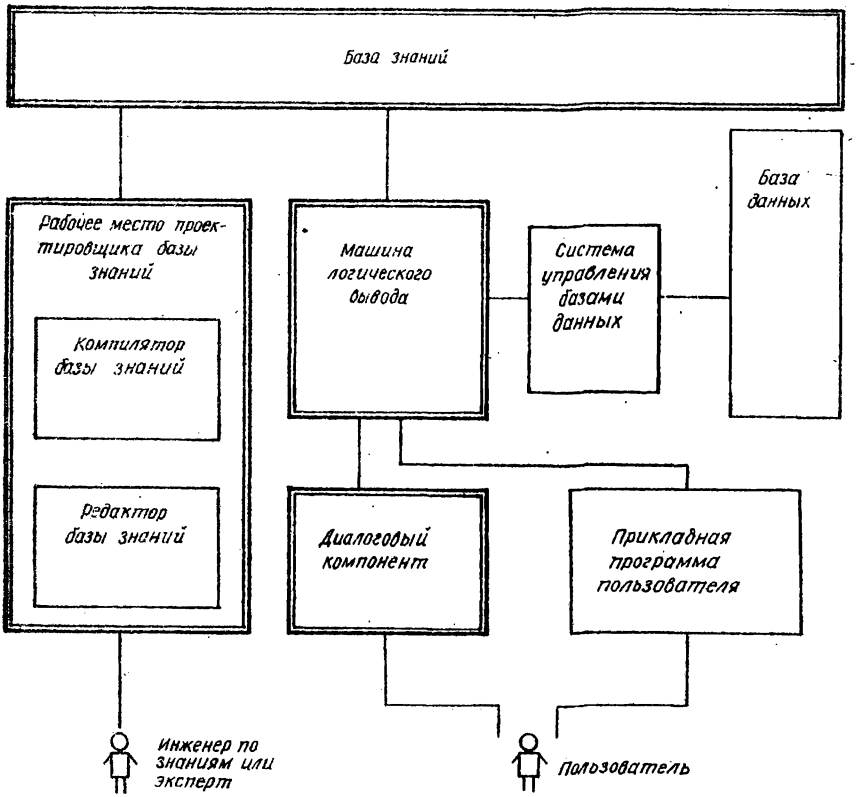



*Рис. 3.2.* Архитектура системы ЭКСПО

ны для хранения знаний. Описатели служат лишь удобными формами словесного описания знаний, хранится же информация совершенно иначе, в частности, структуры хранения ориентированы на применение средств логики предикатов первого порядка и языка Пролог. Трансляцию описателей в структуры хранения обеспечивает компилятор, но для сохранения общности вплоть до главы 7 можно считать, что описатели объектов и правил то же, что обычные фреймы и правила, только названные несколько иначе.

**Архитектура системы.** Основные компоненты инструментального комплекса программ системы ЭКСПО и способы их взаимодействия приведены на рис. 3.2. База знаний располагается во внешней памяти ЭВМ в виде одного или нескольких файлов и ее размер ограничен лишь доступным объемом внешних накопителей. Даже для обычного персонального компьютера это число составляет 40, 80, а иногда и 200 Мбайт. Знания сохраняются во внутренней форме, чтобы обеспечить эффективное запоминание, выборку и обработку описателей. Перевод внешних, ориентированных на пользователя,

элементов знаний (описателей объектов, правил, вопросов и т.д.) во внутреннюю форму обеспечивает компилятор. Одна его часть предназначена собственно для трансляции описателей из внешних форм во внутренние, а другая делает прямо противоположную работу. Обратный перевод внутренних кодов в понятные человеку описатели необходим при корректировке базы знаний, выводе знаний на печать и в некоторых других случаях.

Создание и изменение базы знаний осуществляет инженер по знаниям или непосредственно эксперт. Для этого им дается удобная диалоговая программа, взаимодействующая с инженером или экспертом посредством многочисленных меню, что облегчает создание, корректировку, удаление и компиляцию описателей. Для ввода и изменения описателей используется встроенный текстовый редактор, возможности которого во многом соответствуют функциям таких известных программ, как WORDSTAR, MSWORD или MULTIEDIT. При желании описатели можно подготовить любой программой обработки текстов, а затем прочитать полученный файл внутрь встроенного редактора для последующей компиляции.

Когда возникает необходимость хранить в базе знаний большое количество фактов, к инструментальному комплексу программ можно подключить систему управления базами данных. База знаний с базой данных соединяется посредством описателей вопросов или объектов. Существенно, что архитектура комплекса ЭКСПО не накладывает ограничений на тип используемой системы управления базами данных. Ею может быть реляционная, сетевая или любая иная система. Выбор в каждом конкретном случае определяется желанием пользователя и возможностями той ЭВМ, на которой работает интеллектуальная система. Однако для тесного сопряжения баз знаний и баз данных, осуществляемого посредством описателей объектов (гл. 6), необходимы реляционные средства управления базами данных, поддерживающие язык манипулирования данными SQL. Возможности других средств и языков слишком просты, чтобы с их помощью можно было организовать эффективное хранение и обработку сложных описателей объектов. Однако для доступа к отдельным фактам, управляемого описателями вопросов, подходят любые программы хранения и выборки данных, даже написанные пользователями самостоятельно.

Машина логического вывода осуществляет собственно решение проблем. Результаты решения она передает либо диалоговому компоненту, взаимодействующему с пользователем на понятном ему языке, либо подключенной к системе прикладной программе, написанной для этой цели пользователем. В последнем случае организация диалога, если он необходим, целиком возлагается на программу пользователя. В случаях, когда прикладные программы не используются, для взаимодействия с пользователем служит стандартный диалоговый компонент системы. С его помощью пользователь работает с уже созданными базами знаний. Он указывает имя базы знаний и выбирает решаемую системой проблему, далее ведет диалог с машиной вывода и получает от нее результаты решения. Диалог с пользователем выглядит как обмен фразами на естественном языке с привлечением всевозмож-

ных окон, меню и графических рисунков. При необходимости компонент может дублировать ход диалога на печатающем устройстве, разворачивая содержимое окон экрана в линейную последовательность вопросов, ответов и сопутствующих данных. Кроме того, диалоговый компонент реализует выдачу пользователю объяснений по запросам КАК, ЗАЧЕМ и т.п. Объяснения помещаются в отдельное окно экрана и не мешают ходу диалога.

Если сравнить компоненты комплекса ЭКСПО с обычной интеллектуальной системой (см. рис. 1.2), то при совпадении общей архитектуры, в основе которой лежат одни и те же принципы построения систем этого класса, существуют некоторые различия. Применительно к комплексу ЭКСПО одни компоненты конкретизированы и детализированы, как накопление и изменение знаний, а другие, представляющие в данном случае меньший интерес, объединены в один, как диалог и объяснения. Кроме того, появился ряд новых элементов архитектуры: база данных, система управления ею и прикладная программа. Они указывают на способ объединения средств обработки знаний с другими программными системами, используемый данным инструментальным средством.

## 3.2. ОПИСАНИЕ ОБЪЕКТОВ ПРЕДМЕТНОЙ ОБЛАСТИ

Пользователь начинает построение интеллектуальной системы с описания тех объектов предметной области, информацию о которых он хочет занести в базу знаний и которые понадобятся для решения проблемы. Понятие объекта трактуется достаточно широко. Объекты могут быть: конкретными (как, например, ваш компьютер или именно эта книга) либо абстрактными (множество компьютеров, все книги по базам знаний); реальными либо вымышленными; примитивными (как болт, гайка или другая неделимая деталь) либо составными (из множества деталей состоит, к примеру, компьютер).

Каждому представляющему интерес объекту пользователь присваивает уникальное имя и составляет перечень характеристик, атрибутов объекта. После этого выясняет значения всех атрибутов и для каждого объекта готовит отдельный описатель, ку. ı входят все атрибуты одного объекта. Описатели объектов делятся на два вида — с помощью одних описывают классы объектов, а другие служат для представления информации об элементах этих классов.

**Элементы классов.** Значения любого атрибута можно задать с помощью одной или нескольких констант (рис. 3.3). Это наиболее простой способ присваивания. По умолчанию для любого атрибута допускается множество значений. Если количество значений должно быть строго ограниченным, то следует указать тип атрибута, например:

Атрибут: изготовитель
Тип: однозначный.
Значение: ПО им. Королева, г. Киев.
Объяснение: "Предприятие-изготовитель".

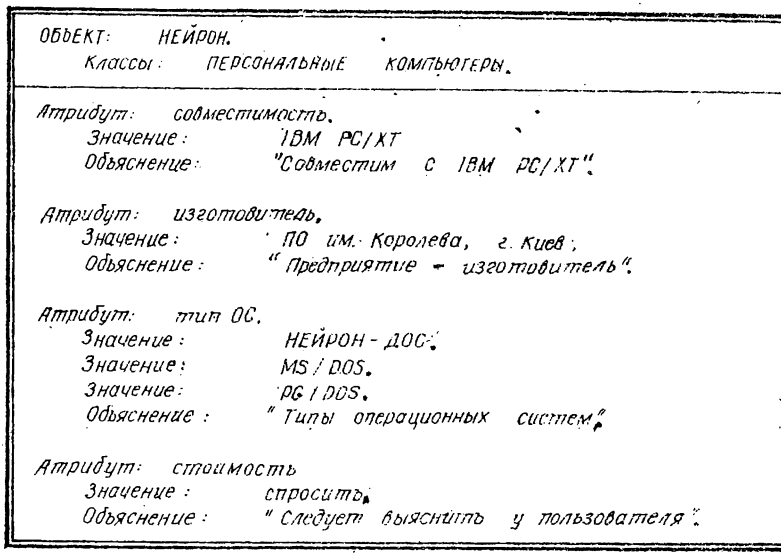



*Рис. 3.3.* Описатель элемента НЕЙРОН из класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ

Атрибут: тип ОС.
Тип: многозначный (5).
Значение: НЕЙРОН–ДОС.
Значение: MS/DOS.
Значение: PC/DOS.
Объяснение: "Типы операционных систем".

В данном случае компьютер НЕЙРОН выпускается только одним изготовителем, т.е. предполагается, что другие производители обязаны использовать иную торговую марку. Напротив, один и тот же компьютер может иметь до пяти типов операционных систем, три из которых заранее известны и заданы константами, еще два могут быть выяснены и добавлены позднее, например в ходе решения некоторой задачи.

В системе принят такой способ описания атрибутов, при котором любую фразу описания, кроме имени атрибута, можно укорачивать и даже вообще опускать. Вместо пропущенных слов компилятор автоматически подставит текст, принятый по умолчанию и запомнит его в базе знаний. Поэтому, к примеру, запись

Атрибут: стоимость.
Тип: многозначный.
Значение: неизвестно.
Объяснение:

равносильна единственной фразе

Атрибут: стоимость.

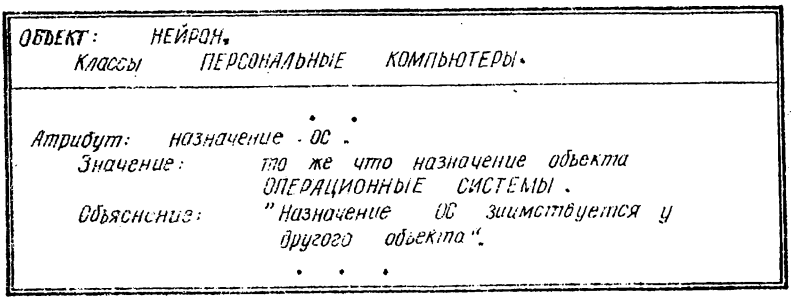



*Рис. 3.4.* Пример заимствования значений

Когда значение какого-либо атрибута следует запросить у пользователя, то этому атрибуту присваивают фиктивную константу "спросить". Константа известна системе, ее появление означает, что для выяснения действительных значений машине вывода необходимо обратиться к соответствующему описателю вопроса и выполнить предписанные там действия, например выдать на экран дисплея текст вопроса и прочитать с клавиатуры ответ пользователя. Затем ответ помещается в рабочую область базы знаний, где находятся все факты, самостоятельно найденные машиной вывода в ходе решения проблемы, и используется наравне с другими фактами так, как будто все они заданы константами.

При необходимости значения атрибута одного объекта заимствуются у другого с помощью фразы "то же что". Например, пусть в описатель объекта НЕЙРОН следует добавить еще один атрибут, описывающий назначение операционных систем компьютера (рис. 3.4). В этом случае заимствование означает, что где-то в базе знаний существует объект ОПЕРАЦИОННЫЕ СИСТЕМЫ, имеющий атрибут "назначение". Все значения этого атрибута присваиваются атрибуту "назначение ОС" объекта НЕЙРОН. С помощью фразы "то же что" можно задать несколько разных значений, заимствуя их у различных объектов, например:

Атрибут: неисправность компьютера.
Значение: то же что неисправность объекта МОНИТОР.
Значение: то же что неисправность объекта ПРОЦЕССОР.
Значение: то же что неисправность объекта КЛАВИАТУРА.
Объяснение: "Неисправность компьютера заимствуется у объектов МОНИТОР, ПРОЦЕССОР и КЛАВИАТУРА".

Значения любого атрибута можно заимствовать не только у одного или нескольких объектов, но и у переменных, которые применяются вместе с правилами логического вывода и рассмотрены вместе с ними далее.

Описатель содержит указание на класс, к которому принадлежит элемент. Одному и тому же элементу разрешается входить в несколько классов, и все они должны быть перечислены сразу после имени объекта. В данном случае объект НЕЙРОН входит лишь в один класс — класс персональных

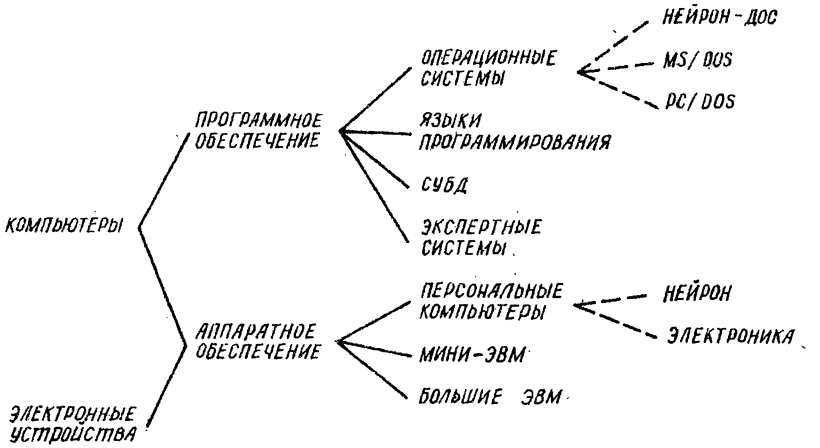



*Рис. 3.5.* База знаний о компьютерах

компьютеров. Появление имени класса предполагает, что в базе знаний существует объект ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ, описывающий этот класс. Принадлежность элементов к классу задается с помощью фразы, начинающейся со слова "Классы:". Если элемент входит в несколько классов, то все они перечисляются в этой фразе через запятую.

**Классы объектов.** В качестве примера на рис. 3.5. приведен фрагмент базы знаний о компьютерах и их программном обеспечении, где имеется ряд связанных друг с другом объектов. Это так называемые таксономические связи, соответствующие отношениям класс — подкласс и класс — элемент, сложившимся между объектами в предметной области. Связи между классами, показанные сплошной линией, означают, что один класс входит как подкласс в некоторый более общий класс. Так, АППАРАТНОЕ ОБЕСПЕЧЕНИЕ является подклассом по отношению к классам ЭЛЕКТРОННЫЕ УСТРОЙСТВА и КОМПЬЮТЕРЫ. Этот тип связи означает, что аппаратура входит в состав компьютера и является электронным устройством. Другая разновидность связи, отмеченная штриховой линией, соответствует отношению класс — элемент и указывает, что некоторый объект, например компьютер НЕЙРОН, является элементом класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ. Оба типа таксономических связей служат для представления статических отношений между объектами предметной области, а также для определения терминов, на которые могут ссылаться другие описатели базы знаний.

Объекты, принадлежащие к одному и тому же классу, имеют общие атрибуты. Эти атрибуты, характеризующие каждый член класса (элемент или подкласс), могут присутствовать в описателе класса наряду с атрибутами, определяющими класс в целом (рис. 3.6). Так, члены класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ имеют четыре общих атрибута, где указываются наименование компьютера, его совместимость, предприятие—изготовитель

```
ОБЪЕКТ: ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ.
    Надклассы:    АППАРАТНОЕ ОБЕСПЕЧЕНИЕ.
    Подклассы:    нет.

Атрибут: наименование.
    Значение:     то же что наименование объекта *.
    Объяснение:   "Наименование компьютера".

Атрибут: совместимость.
    Значение:     то же что совместимость объекта *.
    Объяснение:   "Тип совместимого компьютера".

Атрибут: изготовитель.
    Значение:     то же что изготовитель объекта *.
    Объяснение:   "Предприятие - изготовитель".

Атрибут: тип ОС.
    Значение:     то же что тип ОС объекта *.
    Объяснение:   "Типы операционных систем".

Атрибут: стоимость.
    Значение:     то же что стоимость объекта *.
    Объяснение:   "Стоимость компьютера в рублях".

Атрибут: мин стоимость.
    Значение:     7500.
    Объяснение.   "Минимальная стоимость в рублях".

Атрибут: макс стоимость.
    Значение:     280000.
    Объяснение:   "Максимальная стоимость в рублях".
```

***Рис. 3.6.*** Описатель класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ

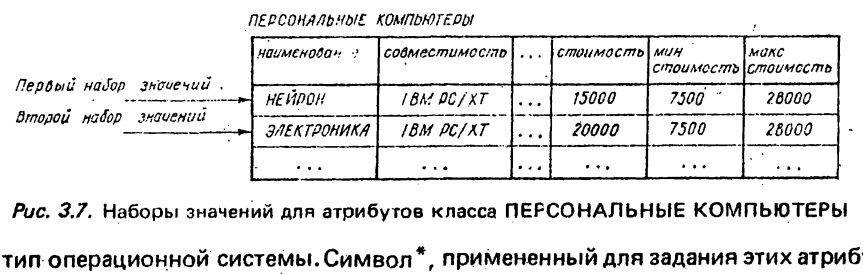



***Рис. 3.7.*** Наборы значений для атрибутов класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ

и тип операционной системы. Символ *, примененный для задания этих атрибутов, означает, что действительные значения должны заимствоваться у всех членов класса, в данном случае — у всех без исключения элементов класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ. Один набор значений этих четырех атрибутов соответствует компьютеру НЕЙРОН, другой — компьютеру ЭЛЕКТРОНИКА, третий — следующему элементу класса и т.д. Два последних атрибута объекта,

задающие минимальную и максимальную стоимости персонального компьютера, напротив, описывают класс в целом. Их значения установлены константами и не изменяются при переборе значений остальных атрибутов. Описатель класса — это как бы окно, через которое в каждый момент видна информация, относящаяся только к одному члену класса (элементу или подклассу). Уместна и другая аналогия: если представить, что информация о всех членах класса имеет вид таблицы, то в каждый момент открыта только одна ее строка (рис. 3.7). Иными словами, всякое упоминание имени класса равносильно перечислению всех членов этого класса.

После того как описатели всех классов и всех элементов классов помещены в базу знаний, из других описателей можно ссылаться на запомненную информацию, выбирая наиболее удобный способ доступа. Например, если возникает необходимость отыскать стоимость компьютера НЕЙРОН, то это можно сделать, запросив либо

стоимость объекта НЕЙРОН,

либо

стоимость объекта ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ,

где

наименование объекта ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ — НЕЙРОН.

Если необходимо сослаться на наименования всех персональных компьютеров, имеющих максимальную стоимость, то записываем.

наименование объекта ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ,

где

стоимость объекта ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ =
= макс стоимость объекта ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ.

Такого рода ссылки на объекты базы знаний часто встречаются в правилах логического вывода. Знак "—" (тире) имеет тот же смысл, что и "=". При формулировании условия можно указывать любой из этих знаков. Математический символ равенства чаще используют при записи арифметических выражений, а тире — при сравнении атрибутов с константами или с другими атрибутами.

Надклассы, куда входит некоторый класс, и подклассы, на которые он делится, указывают в соответствующих фразах, следующих в описателе непосредственно за именем объекта. Когда класс связан с несколькими над- и подклассами, то все они перечисляются в этих фразах через запятую.

**Надежность.** Любому атрибуту разрешается иметь одно или несколько значений. С каждым значением, когда оно запоминается в базе знаний, вс г-да связаны два числа — $P_{min}$ и $P_{max}$, служащих для измерения надежности того, что атрибут действительно имеет указанное значение. Это так называемые субъективные вероятности, изменяющиеся в интервале [0, +1]. Утверждение, что атрибут имеет некоторое значение, — это один факт. Числа $P_{min}$ и $P_{max}$ ($P_{min} \leqslant P_{max}$) показывают: 1) какова вероятность данного факта, 2) какова степень неопределенности для данного факта (рис. 3.8). Предполагается, что точная степень надежности любого факта неизвестна,

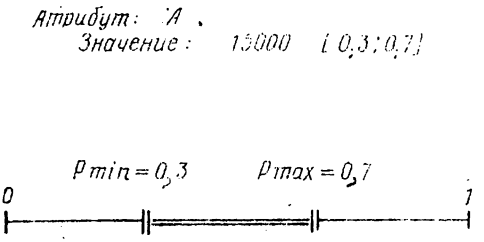



*Рис. 3.8.* Измерение надежности факта

однако вероятность того, что этот факт действительно имеет место, не ниже $P_{min} = 0{,}3$, но и не выше $P_{max} = 0{,}7$, а разность $P_{max} - P_{min} = 0{,}7 - 0{,}3 = 0{,}4$ характеризует неопределенность. Вероятность того, что атрибут имеет указанное значение, лежит в интервале $[P_{min}, P_{max}]$. Величина этого интервала (от 0 до 1), вычисляемая как $P_{max} - P_{min}$, характеризует уровень неопределенности — чем интервал шире, тем менее надежны знания о вероятности и, наоборот, чем он уже, тем больше известно, какова в действительности вероятность данного факта.

В ситуации, когда $P_{min} = P_{max}$, неопределенность отсутствует. Случай $P_{min} = 0$, $P_{max} = 1$, наоборот, соответствует полной неопределенности: известно только то, что вероятность факта лежит между 0 и 1, т.е. ничего не известно. Такому атрибуту в базе знаний присваивается специальное значение "неизвестно". Точнее говоря, любая величина, как только для нее обнаружен интервал [0,1], заменяется специальным значением "неизвестно", которое распознается машиной вывода и обрабатывается особым образом.

Помещая в описатель значение любого атрибута, для него можно указать надежность. Например.

Атрибут: А.
Значение: 15000 [0,3; 0,5].
Значение: 20000 [0,5; 0,7].

означает, что некоторый атрибут А принимает значения 15000 с интервалом вероятности [0,3; 0,5] и 20000 — с интервалом [0,5; 0,7]. По умолчанию предполагается, что каждое значение, когда иного не указано, имеет интервал [1; 1], т.е. вероятность равна единице, а неопределенность отсутствует. Если надежность указана единственным числом, например,

Значение: 20000 [0,7],

то предполагаются отсутствие неопределенности и интервал [0,7; 0,7]. Две фразы

Значение: неизвестно.
и
Значение: < любая величина > [0; 1]

имеют одинаковый смысл.

Независимо от того, каким образом указана надежность в описателе, в базе знаний каждое значение обязательно хранится и обрабатывается вместе с обоими числами $P_{min}$ и $P_{max}$. Именно из них в конечном счете выводится надежность результата решения проблемы.

**Базовые дедуктивные возможности.** Язык описания классов и элементов вместе с поддерживающим его механизмом обработки описателей предназначен для представления в базе знаний информации об объектах предметной области и статических связей между объектами, а также для определения терминов, используемых в базе знаний. Обычно это те же объекты, связи и термины, которые встречаются в предметной области. Описания классов и элементов служат базисом для применения правил логического вывода. Их иногда называют "базой данных" интеллектуальной системы, поскольку используют для представления в базе знаний тех объектов, на которые ссылаются правила вывода [FIK85].

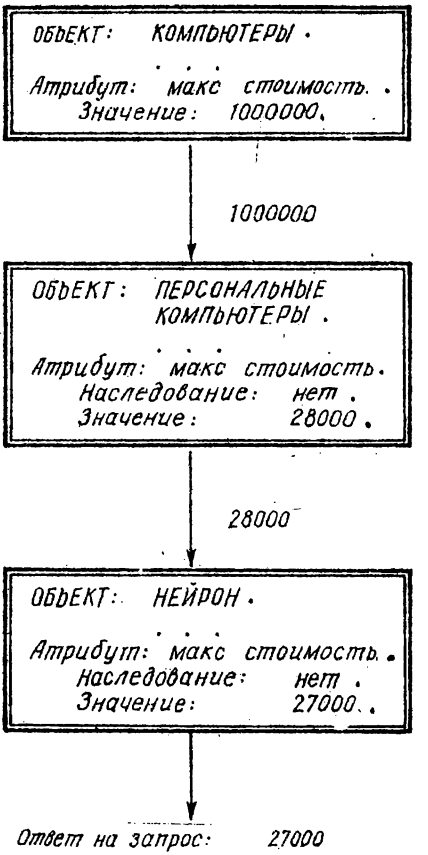



*Рис. 3.9.* Передача значений от класса к подклассу и элементу (наследование не происходит)

Однако средства описания объектов служат не только для хранения информации. С их помощью поддерживается базовый уровень дедукции, который либо невозможно, либо нецелесообразно поддерживать с помощью правил логического вывода. Для этой дедукции пользователю нет необходимости описывать какие-либо специальные правила вывода, так как ее поддерживают другие механизмы, которым не нужны дополнительные правила. Перечень этих механизмов включает: определение терминов; наследование свойств; применение демонов и присоединенных процедур; описание вопросов.

Наиболее простая часть базовых средств логического вывода связана с определениями терминов как классов объектов. Например, если системе предоставлено описание базы знаний о компьютерах (см. рис. 3.5), она может определить, что НЕЙРОН является персональным компьютером — аппаратным обеспечением ЭВМ — компьютером — электронным устройством. Для прослеживания подобных цепочек связей никакие специальные правила логического вывода не нужны. В этом случае поведение системы напо

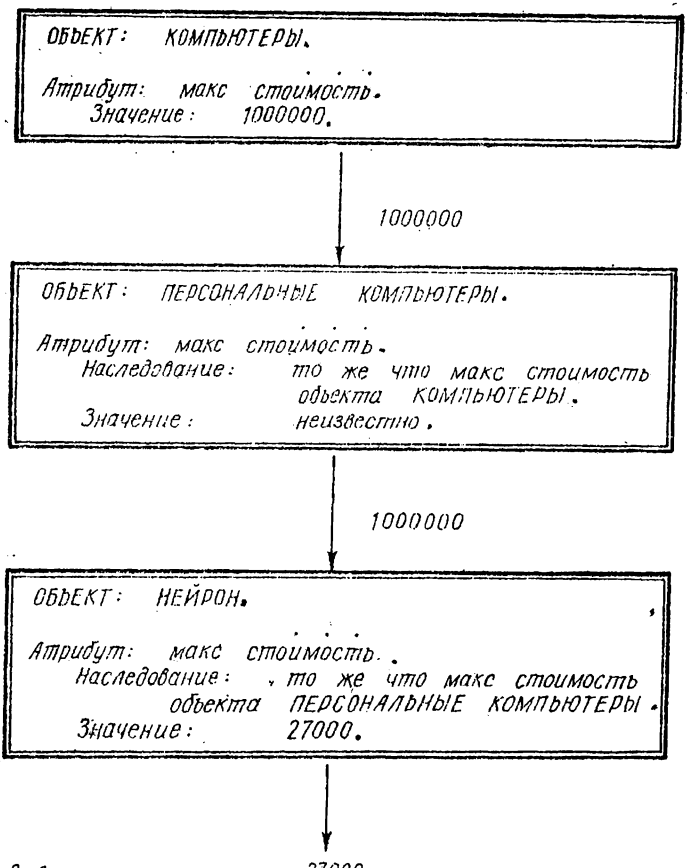



*Рис. 3.10.* Объект ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ наследует значение, а объект НЕЙРОН — нет

чинает обработку семантической сети, в которой также не используются правила вывода.

**Наследование свойств.** Следующая часть дедуктивных возможностей, реализуемых без привлечения правил логического вывода, основана на наследовании свойств, означающем, что члены любого класса могут заимствовать атрибуты, их значения и надежности из описателя своего класса. При этом пользователю разрешается указывать для каждого атрибута тот способ наследования значений у стоящих выше классов, который больше всего отвечает специфике решаемой проблемы.

Например, в описателе объекта НЕЙРОН присутствует атрибут, где указана минимальная стоимость этого компьютера. Тот же атрибут, но объекта ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ имеет другое значение (рис. 3.9.). Когда

наследования "нет", то заимствование значений от класса к подклассу или к элементу не выполняется и каждый объект сохраняет свою собственную стоимость. Этот способ наследования принят в системе по умолчанию, поэтому фразу с указанием на него при описании атрибутов можно опускать.

Другой тип наследования задают с помощью слов "то же что" (рис. 3.10). В фразе, где встречаются эти слова, указывают имя объекта и атрибут, у которого берут значения. Благодаря такому способу наследования быстро отыскивается значение атрибута и его надежность, что заимствуется у объекта стоящего выше уровня. Однако, если значение атрибута уже выяснено, никакого наследования не происходит.

**Демоны.** Значения всех атрибутов, заданных не константами, вычисляются по принципу "когда понадобится". Когда необходимое значение найдено, оно помещается в рабочую область базы знаний и используется, будто было задано константой. С каждым атрибутом разрешается связывать так называемый демон — процедуру, запускаемую в момент, когда добавлено, изменено или удалено значение атрибута. Соответственно и демоны делят на "когда добавлено", "когда изменено" и "когда удалено".

Алгоритм, запускаемый при выполнении любого из перечисленных выше условий, имеет в базе знаний вид отдельной задачи. По принятым в системе соглашениям проблемы складываются из многих частных задач. Одни задачи обрабатываются последовательно, по мере приближения системы к результату решения проблемы, другие — приостанавливают обычную последовательность рассуждений и в паузе выполняют некоторую работу, после чего прерванный ход дедукции возобновляется. Демон предполагает прерывание обычного хода рассуждений и выполнение задачи, имя которой указано в описании атрибута. Например:

Атрибут: стоимость.

| | |
|---|---|
| Когда добавлено: | решить задачу ДОБАВЛЕНА СТОИМОСТЬ. |
| Когда изменено: | решить задачу ИЗМЕНЕНА СТОИМОСТЬ. |
| Когда удалено: | решить задачу УДАЛЕНА СТОИМОСТЬ. |
| Значение: | неизвестно. |

Каждая задача имеет в базе знаний отдельный описатель, содержащий алгоритм ее решения. Решив эту задачу, т.е. выполнив все действия, указанные в ее описателе, машина вывода прекращает обработку демона и возвращается к прерванным рассуждениям. Условие запуска задачи определяется типом демона.

Демон "когда добавлено" означает: если системе станет известно любое новое значение стоимости, она должна занести его в рабочую область базы знаний, а затем решить задачу ДОБАВЛЕНА СТОИМОСТЬ. Предполагается, что в базе знаний имеется описатель задачи с указанным именем, который содержит алгоритм ее решения. Решив эту задачу в соответствии с описателем, машина вывода возобновит прерванные рассуждения. Такая же схема выполнения применяется для демонов "когда изменено" и "когда

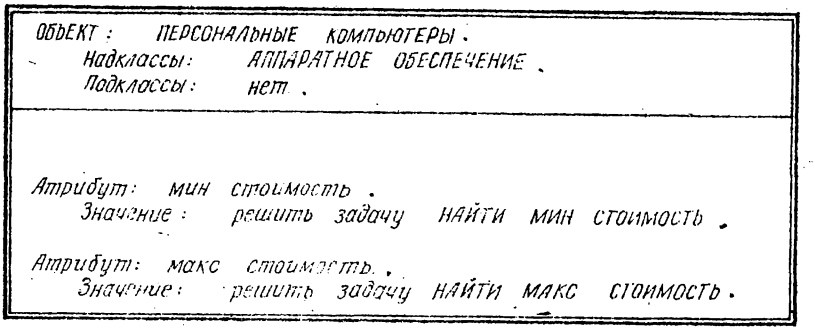



*Рис. 3.11.* Нижняя часть описателя класса, использующего присоединенные процедуры

удалено", однако условием из запуска служит соответственно изменение или удаление одного (любого) значения атрибута.

По умолчанию с атрибутом не связан никакой демон, поэтому ненужные виды демонов разрешается не указывать. Демоны помогают отслеживать изменения значений любого атрибута. Они оказывают помощь, когда вслед за изменением некоторых величин необходимо выполнить какие-либо действия, например обновить изображение на экране дисплея.

**Присоединенные процедуры.** Если для поиска значений атрибута следует выполнить длинную и следовательность операций, например по сложному алгоритму обработать десятки правил логического вывода, то значение атрибута удобно задать, указывая имя задачи, в результате решения которой будет найдена требуемая величина. Алгоритм решения такой задачи образует присоединенную процедуру. Он соединяется с именем атрибута, становясь значением атрибута, и выполняется всякий раз, когда требуется отыскать нужную величину.

Способ подключения присоединенных процедур для вычисления минимальной и максимальной стоимостей персонального компьютера приведен на рис. 3.11. Каждая процедура, являясь обычной задачей, имеет в базе знаний свой описатель, в котором содержится алгоритм ее выполнения. Например, нужно просмотреть все элементы класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ и выбрать в одном случае наименьшее, а в другом — наибольшее значение атрибута "стоимость".

Присоединенные процедуры работают так же, как демоны, — приостанавливают ход рассуждений, выполняют некоторую работу и возобновляют дедукцию. Главное отличие между ними состоит в том, что процедура считается значением атрибута и может наследоваться и заимствоваться с помощью слов "то же что", тогда как демон работает в пределах только того объекта, к которому приписан.

Демоны и присоединенные процедуры, допуская отклонения от регулярной последовательности рассуждений, полезны для решения целого ряда

проблем. Таким способом удается проще и точнее описать желаемый ход дедукции или действия, которые выполняет в аналогичной ситуации человек. В целом демоны и процедуры — это инструмент процедурного подхода к построению баз знаний со всеми присущими ему преимуществами и недостатками. К числу преимуществ относят легкость описания вывода, особенно для проблем, изначально имеющих процедурный характер. Среди недостатков часто называют сложность понимания базы знаний, затрудняющую ее отладку и изменения [БАК92, HAR85]. Иными словами, процедурный подход хорош для проблем, которые легко описать в терминах "как сделать". Декларативные спецификации, напротив, отвечают на вопрос "что истинно", их иногда сложно сформулировать, но зато потом они проще, понятнее и надежнее. Они строятся путем констатации значений атрибутов, отношений между объектами, правил вывода и т.д. Констатируя, что истинно в предметной области, пользователь перекладывает решение вопросов "как сделать" на систему, тогда как при процедурном подходе он обязан разработать детальный алгоритм решения каждой задачи. И хотя многие предпочитают декларативный стиль, инструментальный комплекс ЭКСПО допускает использование обоих подходов либо раздельно, либо, сочетая их в одной и той же базе знаний.

## 3.3. ВОПРОСЫ К ПОЛЬЗОВАТЕЛЮ

Значения некоторых атрибутов, необходимых для решения проблемы, заранее бывают неизвестны, их выясняют у пользователя непосредственно в ходе решения проблемы, задавая ему соответствующий вопрос и читая в базу знаний текст ответа. Таким атрибутам присваивают специальное значение "спросить". Например, для интеллектуальной системы, обслуживающей поиск неисправностей компьютеров, а также оформление платежных документов за работы по ее устранению, представляет интерес срок службы каждой ЭВМ (рис. 3.12). Предполагается, что при сроке службы, не превышающем 1 года, речь идет о гарантийном ремонте, который делается бесплатно. Далее от 1 года до 3 лет применяется льготный послегарантийный тариф, а по истечении этого срока неисправность устраняется за полную стоимость.

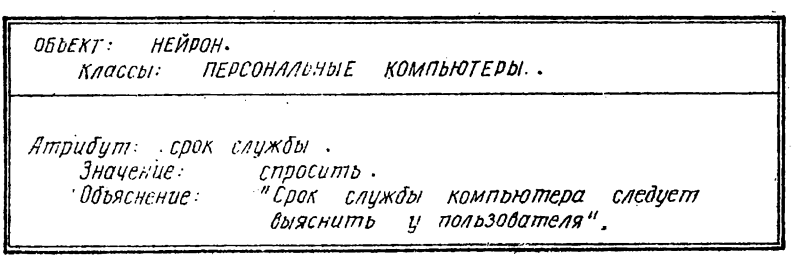



*Рис. 3.12.* Атрибут, значение которого запрашивается у пользователя

```
ВОПРОС:   В-001 .
Объект:   НЕЙРОН .
Атрибут:  срок службы .
Текст:    "Каков .срок службы компьютера (в годах)?".
Справка:  " Укажите, сколько лет эксплуатируется данный компьютер. Это
          может быть целое число или десятичное с точкой , не превышающее 20
          (например, 2,5)"
Значение: число 1 (0,20) .
```

*Рис. 3.13.* Описатель вопроса о сроке службы компьютера

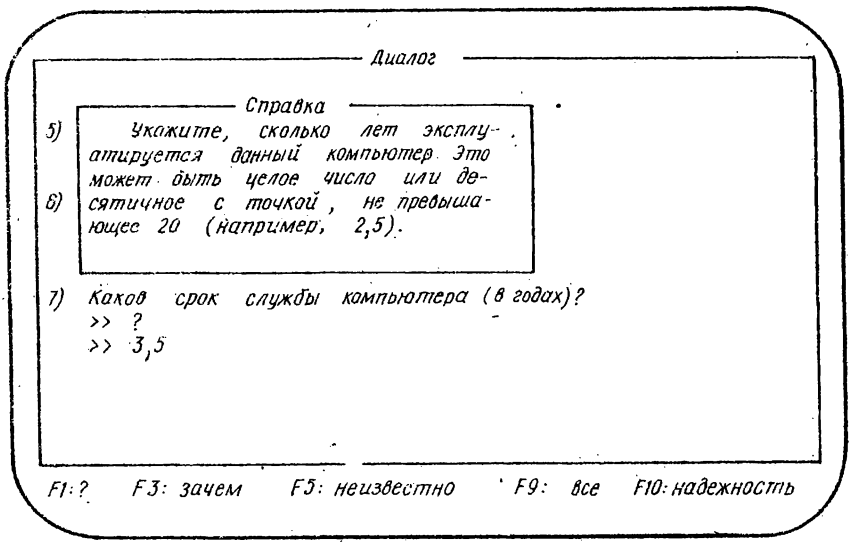



*Рис. 3.14.* Экран дисплея в момент выдачи вопроса о сроке службы компьютера

**Диалог с пользователем.** Наиболее простой вариант описателя вопроса приведен на рис. 3.13, где указаны идентификатор вопроса, объект, а также атрибут, чье значение выясняется. Далее следуют текст вопроса, выдаваемый на экран дисплея, и подсказка, которую можно получить в отдельном окне экрана, отвечая вопросом на вопрос, т.е. знаком вопроса "?" на выведенный текст. Возможное содержимое окон дисплея приведено на рис. 3.14.

Когда пользователь наберет на клавиатуре ответ, выполняется синтаксический контроль поступившего текста. Для этого используется указанная

в описателе область допустимых значений ответа. Запись "число 1 (0,20)" означает, что разрешено только одно действительное (не обязательно целое) число из интервала [0,20]. Любые другие значения, например отрицательные, отвергаются системой. При необходимости можно задать и другие способы синтаксического контроля, например,

| | |
|---|---|
| число (0,20) | — допускается более одного действительного числа из интервала [0; 20]; |
| целое (–1,3) | — разрешены только целые числа –1, 0, 1, 2 или 3; |
| целое1 (–1, 3) | — можно ввести лишь одно целое число из указанного интервала. |

Если ответ на вопрос включает несколько значений, то в конце перечня ставится слово "все". Кроме того, для любого значения можно указать надежность.

>> 3,5 [0; 0,5]
>> 2,5 [0,3]
>> все.

**Простые дедуктивные возможности.** С помощью описателей вопросов реализованы несложные дедуктивные возможности, для поддержания которых не нужно задавать в базе знаний какие-либо правила логического вывода. Например, пользователю разрешается вводить слово "неизвестно" в ответ на любой вопрос. Такой ответ считается вполне допустимым при любом способе синтаксического котроля и помещается в рабочую область базы знаний, как любое другое значение атрибута. Однако в некоторых случаях возникает необходимость, получив ответ "неизвестно", запомнить в базе знаний не его, а некоторое обычно предполагаемое значение. Так, если срок службы компьютера неизвестен, то при составлении платежных документов его можно считать равным 20 годам (рис. 3.15). Обычно предполагаемым значениям, как правило, присваивают низкую надежность (хотя в данном случае это не существенно). Теперь, когда пользователь ответит "неизвестно", в базу знаний вместо этого значения помещается число 20 вместе с той надежностью, что указана в описателе. Заметим, что этот же эффект можно получить с помощью наследования срока службы у любого описателя класса, где задана обычно предполагаемая величина.

```
ВОПРОС:    В-001.
Объект:    НЕЙРОН.
Атрибут:   срок службы.
Текст:
Справка:   "..." .
Значение:  число 1 (0;20).
Обычно:    20 [0;0,5] .
```

*Рис. 3.15.* Описатель вопроса, имеющий обычно предполагаемое значение

Следующий способ несложной дедукции также основан на том, что пользователь дает один ответ на вопрос, а в базу знаний помещается совсем иное значение, на-

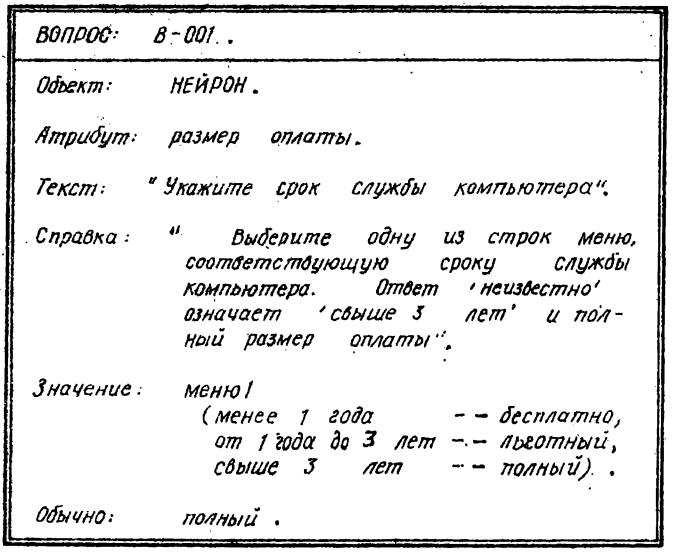



*Рис. 3.16.* Описатель вопроса, предусматривающий преобразование ответа

пример такое, с каким удобнее работать проектировщику базы знаний. Так, в описателе объекта НЕЙРОН атрибут "срок службы" может быть заменен более удобным:

Атрибут: размер оплаты.
Значение: спросить.
Объяснение: "Атрибут принимает значения 'бесплатно', 'льготный' или 'полный' ".

Теперь необходимый описатель вопроса должен предусматривать преобразование данных (рис. 3.16). По-прежнему для любого ответа можно указать надежность, что удобнее сделать, помещая границы интервала в специальное окно экрана (рис. 3..7). Фраза со словом "меню1", означает, что на экране появится соответствующее меню, где разрешается выбрать только одну строку. Пары значений, следующих за этим словом в описателе, задают тексты, которые помещаются в строки меню, и соответствующие им значения атрибута "размер оплаты". Например, когда пользователь выберет вторую строку, то этому атрибуту в базе знаний присваивается значение "льготный".

Другие возможные преобразования ответов задают в описателе с помощью слов:

меню — в этом случае из меню разрешается выбрать более одной строки;

текст 1 — текст единственного ответа вводится с клавиатуры, а затем преобразуется в соответствии с заданными парами значений

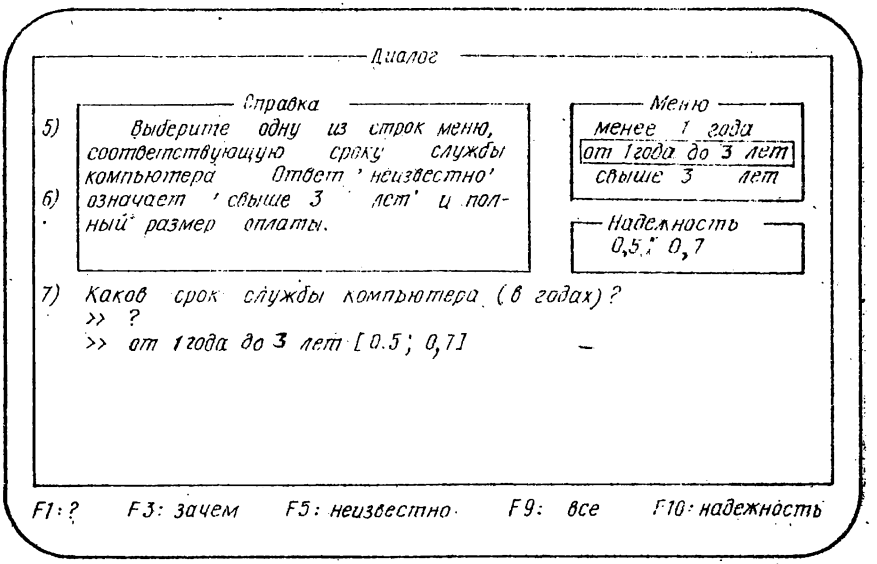



*Рис. 3.17.* Использование меню и преобразований ответа для выяснения размера оплаты

текст — то же, что в предыдущем случае, но допускается несколько вариантов ответа, заканчивающихся словом "все".

Для каждого из этих способов преобразования обязательно задают пары значений, с одними из них работает пользователь, другие запоминаются в базе знаний.

Таким образом, с помощью описателей вопросов не только реализуется поступление в базу знаний информации, полученной от пользователя, но и выполняются несложные операции логического вывода, в основном связанные с преобразованием данных. Те же преобразования можно было бы выполнить с помощью наследования свойств или правил логического вывода. Ниже приведено несколько правил, описывающих все необходимые преобразования для последнего примера:

| | | |
|---|---|---|
| Если | срок службы объекта НЕЙРОН | менее 1 года, |
| то | размер оплаты объекта НЕЙРОН | бесплатно. |
| Если | срок службы объекта НЕЙРОН | от 1 года до 3 лет, |
| то | размер оплаты объекта НЕЙРОН | льготный. |
| Если | срок службы объекта НЕЙРОН | свыше 3 лет, |
| то | размер оплаты объекта НЕЙРОН | полный. |
| Если | срок службы объекта НЕЙРОН | неизвестен, |
| то | размер оплаты объекта НЕЙРОН | полный. |

Кроме увеличения чисто механической работы по написанию таких конструкций, а также затрат памяти на их хранение и времени на обработку, правила не сообщают системе ничего нового и не увеличивают ее знания по

сравнению с информацией, содержащейся в описателе вопроса. Однако обратное утверждение было бы неверным, так как с помощью правил удается поддерживать возможности дедукции, многие из которых попросту не по силам описателям ни вопросов, ни объектов предметной области.

## 3.4. ПРАВИЛА ЛОГИЧЕСКОГО ВЫВОДА

Описания объектов и вопросов играют роль "базы данных" интеллектуальной системы. С их помощью формализуется наиболее статичная часть знаний о предметной области. Однако даже в этом случае присутствуют такие элементы описания знаний, посредством которых передается динамика логического вывода. Это касается прежде всего средств, поддерживающих базовые дедуктивные возможности: демонов, присоединенных процедур, механизмов наследования свойств, преобразования значений и т.д. Интеллектуальную систему можно построить, ограничиваясь применением только этих базовых средств вывода, имеющихся в описателях объектов и вопросов, но такая система сможет, по-видимому, решать лишь очень простые задачи диагностики и классификации. Для более сложных проблем, составляющих в количественном отношении подавляющее большинство, в базу знаний необходимо вводить специальные правила логического вывода, в соответствии с которыми машина вывода обрабатывает "данные" об объектах и вопросах. Правила — такой же элемент описания предметной области, как информация об объектах и вопросах.

Логика построения базы знаний для системы ЭКСПО такова. Сначала пользователь выявляет в предметной области данные, с которыми работает создаваемая интеллектуальная система, и заносит их в базу знаний с помощью средств описания объектов и вопросов. Затем формулирует правила рассуждений об объектах предметной области и запоминает их в базе знаний, пользуясь описателями правил логического вывода. Позднее интеллектуальная система, обрабатывая правила вывода, нужным образом интерпретирует знания об объектах, и, делая логические заключения, в конце концов отыскивает решение задачи.

Каждое правило олицетворяет один логический переход в пространстве решений проблемы. Множество правил в совокупности описывает все возможные пути в пространстве решений, т.е. все цепочки рассуждений, ведущие от исходных данных к результату. Правило имеет условие и заключение (или вывод). С декларативной точки зрения оно констатирует, что если указанное условие истинно, то таковым следует считать и заключение правила. С процедурной точки зрения правило можно рассматривать как команду машине вывода выполнить определенную работу, в частности проверить истинность условия, после чего поместить в рабочую область указанное в правиле заключение.

Условие правила включает одно или несколько сравнений, соединенных обычными союзами "и", "или", "не" и т.д. Например, когда сравниваются

значения атрибутов одних объектов со значениями атрибутов этих же или других объектов и все сравнения, входящие в условие, оказываются истинными, то машина вывода считает истинным и заключение правила, выполняя подразумеваемые этим заключением действия, например запоминая в базе знаний новое значение для указанного атрибута. Таким образом, с помощью правил из значений одних атрибутов логически выводят значения других.

**Переменные.** Как в условиях, так и в заключениях правил наряду с атрибутами разрешается использовать особые переменные, не имеющие отношения к каким-либо объектам базы знаний. Каждая такая переменная имеет собственное имя, с которым связаны одно или несколько значений. Понятие переменной в системе имеет тот же смысл, что и в обычных языках программирования, т.е. имя переменной служит идентификатором для одного или нескольких значений. Каждое значение переменной сохраняется в базе знаний вместе с числами $P_{min}$ и $P_{max}$, устанавливающими интервал надежности данной величины. Переменные принимают все те же значения, что и атрибуты объектов, и, в частности, могут иметь значение "неизвестно". Например, переменная с именем "состояние компьютера" может принимать два значения — "исправен" и "неисправен". Эти значения устанавливаются в результате обработки следующих правил.

| | | | |
|---|---|---|---|
| Если | | состояние объекта МОНИТОР | неисправен |
| | или | состояние объекта ПРОЦЕССОР | неисправен |
| | или | состояние объекта КЛАВИАТУРА | неисправна, |
| то | | состояние компьютера | неисправен. |
| Если | | состояние объекта МОНИТОР | исправен |
| | и | состояние объекта ПРОЦЕССОР | исправен |
| | и | состояние объекта КЛАВИАТУРА | исправна, |
| то | | состояние компьютера | исправен. |

В данном случае значение переменной выводится на основании анализа атрибутов нескольких различных объектов. Возможен и обратный процесс, когда атрибуту некоторого объекта присваивают значение переменной, например:

| | |
|---|---|
| Атрибут: | состояние. |
| Значение: | то же что состояние компьютера. |
| Объяснение: | "Атрибуту 'состояние' объекта НЕЙРОН присваивается значение переменной, выведенное с помощью правил". |

Переменные можно считать атрибутами, только не приписанными ни к какому объекту. Имена переменных строятся так же, как и имена атрибутов, просто при записи переменной в любом описателе для нее не указывается имя объекта. Более того, переменные и атрибуты в системе синтаксически равноправны. Это означает, что когда в правиле встречаются имена атрибута и объекта, на их месте может стоять имя соответствующей переменной, и наоборот. Таким образом, и заключения правил могут ссылаться

| |
|---|
| ВОПРОС: В-010 . |
| Переменная : срок службы компьютера. |
| Текст: "Каков срок службы компьютера (в годах)?". |
| Справка: " Укажите, сколько лет эксплуатируется данный компьютер. Это может быть целое число или десятичное с точкой, не превышающее 20 (например, 2,5)". |
| Значение: число 1 (0,20). |

*Рис. 3.18.* Описатель вопроса, с помощью которого выясняется значение переменной "срок службы компьютера"

только на атрибуты либо только на переменные, либо и на те и на другие одновременно.

Главное назначение переменных — обслуживать процессы дедукции и прежде всего передачу значений между различными правилами логического вывода, а также от правил к объектам, вопросам и обратно. Приведенные выше примеры вместе и иллюстрируют передачу значений от объектов МОНИТОР, ПРОЦЕССОР и КЛАВИАТУРА через правила вывода к объекту НЕЙРОН.

Когда значение какой-либо переменной необходимо выяснить у пользователя, то следует подготовить описатель вопроса, где вместо имен атрибута и объекта следует указать имя переменной (рис. 3.18). После того как у пользователя выяснено действительное значение переменной, оно запоминается в базе знаний и используется машиной вывода так же, как и любая другая величина из рабочей области.

Переменные служат для увязывания правил в цепочки рассуждений. Значение любой переменной может быть выведено с помощью одного правила, а затем использовано в условии другого. Таким образом, в базе знаний образуются длинные цепочки правил, двигаясь по которым машина вывода проходит путь от исходных данных к результату решения проблемы. Передача значений от правила к правилу осуществляется через рабочую область базы знаний. В одном случае цепочка начинается с анализа атрибутов некоторых объектов базы знаний, в результате чего выясняются значения ряда переменных, которые затем присваиваются атрибутам тех же или других объектов. В другом случае сначала с помощью вопросов выясняются исходные значения одних переменных, из них путем применения многих правил выводятся другие переменные, и они запоминаются как значения атрибутов. В третьем случае цепочка оканчивается не присваиванием значений атрибутам, а выяснением величин нескольких переменных, образующих в совокупности необходимый результат. Например, результатом решения проблемы диагностики

```
ПРАВИЛО:  П-070 .

Переменная:     состояние   компьютера .
  Значение :    неисправен.

   Если:        состояние   объекта  МОНИТОР      -- неисправен
        или     состояние   объекта  ПРОЦЕССОР    -- неисправен
        или     состояние   объекта  КЛАВИАТУРА   -- неисправна.

   Объяснение :
    "Правило:   П-070 .

      Если          <1>  у компьютера неисправен  монитор
              или   <2>  неисправен  процессор
              или   <3>  неисправна  клавиатура,
         то              компьютер   неисправен.

     Автор:  Иванов А.Б.
     Дата:   15.02.91 .
     Источник:  ...". .
```

*Рис. 3.19.* Правило — присваивание одного значения переменной

компьютера могут служить значения двух текстовых переменных — "причина неисправности" и "рекомендации по устранению поломки". Допускаются и многочисленные другие комбинации переменных и атрибутов при построении цепочек правил, а также объединение нескольких цепочек в длинную последовательность рассуждений через совместно используемые переменные или атрибуты объектов. В результате такого объединения обращение, например, за значением какого-то одного атрибута иногда приводит к обработке длинного перечня правил, где задействованы многие переменные и атрибуты нескольких объектов. И только обработав эту последовательность правил, машина вывода определит нужную величину.

**Описатели правил.** Согласно принятым в системе соглашениям каждое правило рассматривается как средство логического вывода одного или нескольких значений либо переменных, либо атрибутов объектов. Поэтому в описателе правила для удобства указываются сначала имена переменных и атрибутов вместе со значениями, а затем необходимое условие. Кроме них в описатель включают различную необязательную служебную информацию — это текст правила на естественном языке, фамилия автора правила, дата последнего изменения описателя и литературный либо иной источник, откуда получено правило.

Внешний вид простейшего описателя правила приведен на рис. 3.19, где переменной "состояние компьютера" в зависимости от истинности условия присваивается значение "неисправен". Описатель содержит также текст правила, который не обязательно совпадает с формальной записью правила. Этот текст выдается пользователю системы в качестве объяснений, когда они запрашиваются при диалоге в ходе решения проблемы. Если пользователю

более понятна запись, где не встречаются ссылки на имена объектов и атрибутов, то их явное упоминание можно опустить. Этот текст не подвергается синтаксическому анализу, он сохраняется в базе знаний отдельно от скомпилированных правил и содержит любую информацию, которая может понадобиться пользователю в ходе решения проблемы, а также создателю базы знаний при ее построении и корректировках. Если текст вместе со словом "Объяснение" в описателе опущен, то вместо него в качестве объяснения пользователю выдается целиком часть описателя, которая расположенная выше пропущенного слова. Такой способ работы предпочитают квалифицированные пользователи. Кроме того, его применяют, когда размер памяти компьютера, выделенной под базу знаний, ограничен.

В одном и том же описателе разрешается присваивать переменной или атрибуту несколько различных значений (рис. 3.20). В таком описателе удобно объединять все значения для одной переменной вместе с необходимыми условиями. Поскольку не накладывается никаких ограничений на выбор идентификатора описателя, то в рассмотренном примере имеет смысл назвать правило не абстрактным номером, а по имени переменной. Этот прием, облегчая понимание смысла описателей, упрощает конструирование базы знаний, ее тестирование и последующие изменения.

Допускается и диаметрально противоположный подход, когда правило имеет одно условие и несколько заключений, соединенных союзом "и". Например, в правиле

| | | | |
|---|---|---|---|
| Если | | состояние объекта ПЛАТА ПРОЦЕССОРА | неисправна |
| | или | состояние объекта ПЛАТА ПАМЯТИ | неисправна, |
| то | | состояние объекта ПРОЦЕССОР | неисправен |
| | и | состояние компьютера | неисправен |

истинность условия связывается с присваиванием значений, во-первых, атрибуту объекта и, во-вторых, переменной (рис. 3.21). Этот пример иллюстрирует также, что вместо имени переменной в описателе правила могут стоять имена объекта и атрибута. Кроме того, каждому значению становится в соответствие свой собственный интервал надежности. Надежность в описателях правил всегда задается с помощью круглых, а не квадратных скобок (см. гл. 5).

В зависимости от того, какой стиль выбрал для себя проектировщик базы знаний, он может строить описатели правил по принципу "одна переменная и все условия" или "одно условие и все заключения". Выбор наиболее подходящего способа зависит от специфики создаваемой базы знаний, т.е. предметной области и задачи, с которыми имеет дело проектировщик. Обычно первый подход особенно удобен для задач, решаемых по методу обратной цепочки рассуждений, а второй — для прямой цепочки. Разрешается объединять в одной базе знаний оба подхода одновременно, используя первый метод для одних задач, а второй — для других. Компилятор описателей допускает объединение в одном правиле нескольких условий с несколькими заключениями, однако не желательно придерживаться такого

```
ПРАВИЛО:  состояние  компьютера.

Переменная:     состояние  компьютера.
  Значение:     неисправен.

     Если:      состояние  объекта  МОНИТОР     -- неисправен
           или  состояние  объекта  ПРОЦЕССОР   -- неисправен
           или  состояние  объекта  КЛАВИАТУРА  -- неисправна.

  Объяснение:
    " Правило:  состояние  компьютера  (А) .

        Если        <1>  у компьютера неисправен монитор
              или   <2>  неисправен  процессор
              или   <3>  неисправна  клавиатура,
          то             компьютер   неисправен.

    Автор:  Иванов А.Б.
    Дата:   15.02.91.
    Источник:     ".

  Значение:  исправен.

     Если:      состояние  объекта  МОНИТОР     -- исправен
           и    состояние  объекта  ПРОЦЕССОР   -- исправен
           и    состояние  объекта  КЛАВИАТУРА  -- исправна.

  Объяснение:
    " Правило:  состояние  компьютера  (Б).

        Если        <1>  у компьютера исправен монитор
              и     <2>  исправен  процессор
              и     <3>  исправна  клавиатура,
          то             компьютер   исправен.

    Автор:  Иванов  А.Б.
    Дата:   15.02.91.
    Источник:  ...  ".
```

***Рис. 3.20.*** **Правило — присваивание двух значений переменной**

```
ПРАВИЛО:  П-032.

Объект:      ПРОЦЕССОР .
Атрибут:     состояние .
  Значение:  неисправен   ( 0,9; 0,98).

Переменная:  состояние компьютера.
  Значение:  неисправен  ( 0,87; 0,92).

     Если:      состояние  объекта  ПЛАТА ПРОЦЕССОРА  -- неисправна
           или  состояние  объекта  ПЛАТА ПАМЯТИ     -- неисправна.
```

***Рис. 3.21.*** **Описатель правила. Два заключения, соединенные союзом "и"**

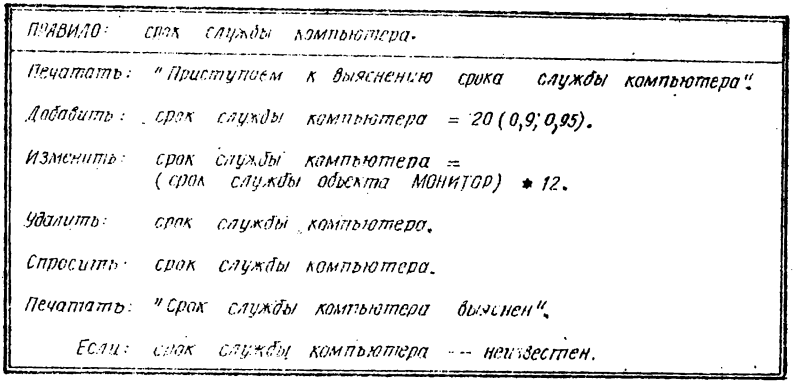



*Рис. 3.22.* Правило с командами

способа записи, поскольку он, как и малосодержательные имена атрибутов, переменных и описателей, усложняет понимание базы знаний и, следовательно, снижает производительность как инженеров по знаниям, так и пользователей интеллектуальной системы.

В целом принципы записи описателей правил просты и минимально ограничивают фантазию проектировщика базы знаний. Он может выбирать любые удобные идентификаторы атрибутов, переменных и правил, строить сложные условия, используя союзы "и", "или", "не", арифметические операции, знаки сравнения величин и скобки в общематематическом смысле, присваивать множество значений нескольким различным атрибутам и переменным, задавать перечни условий "Если" и т.п. Однако необходимо помнить, что заключения правил всегда соединяются только союзом "и", который хотя и не становится в описателях, но всегда подразумевается. Фактически описатели правил построены таким образом, что вставить союз "или" просто некуда.

**Команды.** В правилах разрешается задавать команды, прямо предписывающие машине вывода совершить определенную работу. Большинство команд манипулируют содержимым рабочей области базы знаний, добавляют, удаляют или изменяют находящиеся там значения. Ряд команд предназначен для вывода сообщений на экран и печать, выдачи вопросов и чтения ответов, управления решением задач.

Так, с помощью команды "печатать" на экран дисплея и устройство печати, где отображается диалог с пользователем, выводится указанный текст (рис. 3.22). Команда полезна в тех случаях, когда возникает необходимость проинформировать пользователя о начале решения некоторой подзадачи или о смене режима работы машины вывода, а также если следует поставить пользователя в известность о том, чем именно занята в данный момент система.

Команда "добавить" к существующим, уже известным значениям переменной или атрибута добавляет новые значения. При желании название команды в описателе можно опускать, т.е. заменять запись

Добавить: срок службы компьютера = 20 (0,9; 0,95)
двумя фразами
Переменная: срок службы компьютера
Значение: 20 (0,9; 0,95).

Команды, перечисленные в описателе правила, всегда выполняются строго последовательно сверху вниз и лишь после того, как машина вывода убедится в истинности условия. Поэтому правило, приведенное на рис. 3.22, начнет выполняться только тогда, когда выяснится, что срок службы компьютера системе неизвестен. После этого машина вывода в соответствии с перечнем команд, составляющим заключение правила, сперва напечатает сообщение о начале выполнения правила. Затем заменит в базе знаний неизвестное значение срока службы компьютера величиной 20. Далее по порядку обрабатываются все остальные команды, входящие в заключение.

Команда "изменить" стирает в базе знаний все значения указанной переменной, а затем добавляет туда новую величину. В данном случае из рабочей области изымается значение 20 и вместо него помещается величина, вычисленная как частное от деления срока службы монитора в месяцах на константу 12. Все команды, связанные с добавлением и изменением значений, могут иметь в качестве аргумента арифметическую формулу, по которой рассчитывается новая величина пременной или атрибут.

Каманда "удалить" исключает из базы знаний все значения переменной или атрибута. По результату своих действий пара команд "удалить" и затем "добавить" соответствует одной команде "изменить".

Команда "спросить" заставляет машину вывода обработать описатель вопроса с указанным именем. В данном примере предполагается, что в базе знаний присутствует описатель вопроса с именем "срок службы компьютера", где имеется текст вопроса и другая необходимая информация. После обработки этого описателя в рабочую область заносят полученное от пользователя значение переменной или атрибута.

В условиях правил разрешается включать служебные слова "известно" и "неизвестно", понятные машине вывода и обрабатываемые ею специальным образом. Сравнение атрибута или переменной со значением "известно" считается истинным, если в базе знаний имеется хотя бы одно любое значение для этой переменной или этого атрибута, кроме, естественно, неизвестного. Если в базе знаний нет ни одного значения или присутствует "неизвестно", истинным считается сравнение переменной либо атрибута с неизвестной величиной. С помощью таких специально обрабатываемых величин несложно реализовать присваивание переменным и атрибутам обычно предполагаемых значений. Необходимые для этого правила строятся по единой схеме: если машина вывода пыталась отыскать значение переменной или атрибута, но оно все-таки осталось неизвестным, то следует присвоить переменной или атрибуту обычно предполагаемую величину. Например, правило

Объект МОНИТОР
Атрибут: срок службы.
Значение: (срок службы компьютера) * 12.
Если: срок службы объекта МОНИТОР — — неизвестен.

позволяет присвоить атрибуту "срок службы" объекта МОНИТОР, измеряемому в месяцах работы, значение переменной "срок службы компьютера", когда никаких других данных о периоде службы монитора не имеется.

Еще один важный прием часто используют в правилах, обрабатываемых по методу прямой цепочки рассуждений. При этом заводят отдельную переменную с именем, скажем, "текущий контекст", изменяя единственное значение которой управляют выборкой правил из базы знаний. Например, правило

Печатать: "Неисправность компьютера найдена".
Изменить: текущий контекст = неисправность найдена.
Если: текущий контекст — — поиск неисправности
и неисправность компьютера — — известна.

позволяет переключить машину вывода от поиска неисправности к выработке рекомендаций по устранению поломки компьютера. Если теперь все правила, относящиеся к поиску неисправности, начинать с условия

Если: текущий контекст — — поиск неисправности
и . . . ,

а условиям всех правил, участвующих в выработке рекомендаций по починке, придать вид

Если: текущий контекст — — неисправность найдена
и . . . ,

то, устанавливая нужное значение контекста, можно переключать машину вывода от обработки одного множества правил к обработке другого.

Тот же эффект достигается и другим способом. Для этого подхода нет необходимости заводить вспомогательную переменную и упоминать ее первой строкой во всех правилах логического вывода. Достаточно включить в вывод правила специальную команду запуска задачи. Например:

Решить задачу: выработка рекомендаций.
Если: неисправность компьютера — — известна.

Когда найдена неисправность компьютера, машина вывода приступит к решению задачи с именем "выработка рекомендаций". Эта задача должна иметь в базе знаний отдельный описатель с указанным идентификатором. Решая эту задачу в соответствии с описателем, машина вывода имеет дело только с составляющим данную задачу подмножеством правил.

Оба подхода позволяют сегментировать пространство решений проблемы, добиваясь того, чтобы в каждый момент времени машина вывода "видела" не всю базу знаний сразу, а только ее сегмент (подмножество правил или отдельную задачу), действительно необходимый на текущем шаге. Тем

самым создаются предпосылки для увеличения скорости логического вывода особенно в тех случаях, когда база знаний достаточно велика, к примеру содержит несколько сотен или тысяч правил. Для решения всей проблемы по-прежнему задействованы все необходимые правила, но на каждом шаге рассматривается только малое их подмножество.

**Организация базы знаний.** Правила с командами носят явно выраженный процедурный характер. Они прямо предписывают машине выводы выполнить те или иные действия. Соответственно эти правила имеют структуру

Если < условие > , то < действие > .

Применение правил с командами при построении базы знаний фактически означает, что проектировщик программирует работу машины вывода на специфическом "языке программирования" — языке описания правил.

Многие считают, что процедурные спецификации несколько сложнее декларативных, где никакие команды запоминания и изменения знаний не указываются. Кроме того, они предъявляют более высокие требования к квалификации разработчиков интеллектуальной системы и, немного снижая наглядность базы знаний, усложняют ее построение и особенно отладку.

Декларативные описания правил строятся таким образом, что выполняемые машиной вывода действия в них явно не указываются и даже не обязательно подразумеваются. Эти правила имеют структуру

Если < условие > , то < логическое заключение > .

Составляя такие правила, пользователь просто констатирует, что истинно в предметной области. Он может даже ничего не знать о присваивании значений переменным и атрибутам, о применении рабочей области и т.д. Все это облегчает работу пользователям — непрограммистам и позволяет создавать интеллектуальные системы самостоятельно. Декларативные спецификации облегчают работу и квалифицированных инженеров по знаниям, позволяя им (там, где это возможно) не углубляться в детали манипулирования знаниями.

С помощью процедурных описателей удается точнее настроить инструментальное средство на особенности той или иной предметной области, и эффективнее использовать память и время процессора. Однако трудоемкость этого подхода может превышать затраты на составление декларативных описателей. В то же время во многих предметных областях встречаются проблемы, которым изначально присущ процедурный характер. Эти проблемы словесно обычно описывают в терминах "что делать, если...". Для таких задач применение процедурного подхода является не только естественным способом построения интеллектуальной системы, но и наименее трудоемкой альтернативой, приводящей к созданию простой и наглядной базы знаний.

Инструментальный комплекс программ допускает применение либо процедурного, либо декларативного стиля в отдельности, а также объединение обоих подходов в рамках одной базы знаний (и даже в пределах одного описателя правила). При этом проблему обычно делят на относительно обособленные задачи, а затем используют для каждой задачи подход,

в наибольшей степени отвечающий ее специфике и, следовательно, требующий наименьших усилий для создания базы знаний.

Декларативные и процедурные правила, использующие переменные, представляют собой мощный инструмент описания баз знаний и покрывают многие возможности, свойственные описателям объектов. Поэтому инструментальный комплекс программ допускает создание таких баз знаний, где вообще отсутствуют описатели объектов. Например, по такой схеме можно построить интеллектуальную систему для предметной области, в которой имеется лишь один объект со многими атрибутами. В этой системе заводить единственный описатель объекта, по-видимому, не имеет смысла. В других случаях возникают трудности при выделении в предметной области обособленных объектов либо речь идет о таких вещах, которые вообще трудно отнести к объектам. Во всех этих ситуациях база знаний системы может не содержать ни одного описателя объекта, а решение задачи целиком обеспечивается за счет переменных и правил. В целом наличие в базе знаний описателей вопросов определяется спецификой предметной области и желанием пользователя. Выделение объектов оправдано для тех областей, где это удается сделать без особого труда. И наоборот, если задача проще решается с помощью только переменных и правил, то именно их и следует использовать.

Вне зависимости от того, какой подход выбран для создания интеллектуальной системы, ее база знаний должна обязательно включать не только описатели правил и (или) объектов, но и хотя бы один описатель проблемы и, возможно, несколько описателей задач, на которые делится эта проблема. Назначение, способы построения и обработки этих описателей, а также предоставляемые в этой части возможности инструментального средства детально рассмотрены в следующей главе.

*ГЛАВА 4*

# СПОСОБЫ ЛОГИЧЕСКОГО ВЫВОДА

Процесс построения интеллектуальной системы, способной эффективно решать сложные задачи, последовательно проходит три этапа: 1) выяснение, приобретение знаний, которыми обладают специалисты в выбранной предметной области; 2) формализацию знаний для использования в компьютерной системе; 3) реализацию системы на ЭВМ. Затем создаваемую интеллектуальную систему тестируют, как правило, на реальных примерах, и при выявлении отклонений от желаемого поведения ее разработчики возвращаются на один из предыдущих этапов, обычно к уточнению знаний экспертов; затем к их формализации и реализации новой, уточненной версии системы.

Выяснение знаний специалистов представляет собой весьма непростую задачу, поскольку эксперты не мыслят в терминах инженерии знаний и часто затрудняются изложить последовательность логических заключений, приводящих к тому или иному выводу. Для работы в этих условиях инженеры по знаниям применяют специальные методики общения с экспертами, чтобы выяснить, какие собственно знания необходимы человеку для успешного решения поставленной проблемы и какие методы дедукции применяют эксперты. Такие методические рекомендации обычно представляют собой свод правил поведения инженера во время его встреч с экспертами [ОСУ90, УОТ89].

В последние годы все большее распространение получают специальные программные системы, выявляющие и структурирующие знания экспертов. Самая первая система этого типа — TIMM [HAR85] — фактически сама является интеллектуальной. Она способна, ведя диалог с проектировщиком базы знаний, выяснять у него особенности предметной области и задачи, а затем строить правила логического вывода, составляющие нужную базу знаний. Примерно по такой схеме работают и более поздние системы выявления и накопления знаний — MOLE [ESH86], ROGET[BEN85] и другие [SCH88].

После того как необходимые знания выяснены и должным образом структурированы, их необходимо формализовать, чтобы сделать понятными не только людям, но и компьютерной системе. Попросту говоря, на этом шаге понятные людям словесные формулировки знаний преобразуют в такие конструкции, как фреймы, правила продукций, алгоритмы дедукции и т.п. Применяемый способ формализации сильно зависит от того, какие

формы представления знаний и методы дедукции поддерживает то инструментальное средство, с помощью которого позднее реализуется интеллектуальная система. По аналогии с областью баз данных можно сказать, что инфологическая модель базы знаний (характеризующая информацию как таковую) преобразуется в даталогическую (ориентированную на специфику используемых программных средств).

Пусть для создания интеллектуальной системы выбрано прямое программирование на одном из языков Си, Паскале, Лиспе или Прологе. На каждом из них можно реализовать любые способы представления знаний и методы логического вывода, однако затраты на формализацию будут максимальны, так как конструкции языков программирования наиболее далеки от тех, в терминах которых мыслят и излагают свои знания эксперты.

Если использовать средство окружения или среды, где уже имеются какие-то формы представления знаний и методы дедукции, напоминающие элементы знаний экспертов, то затраты на формализацию сократятся. Однако чем жестче предъявляемые этим инструментальным средством требования, тем больших усилий, вообще говоря, может потребовать формализация знаний. Однако чем мягче требования, тем обычно меньше "умеет" инструментальное средство, а значит, построение интеллектуальной системы с его помощью будет более дорогим и трудоемким делом.

Предельно полярные соотношения трудоемкости реализации и затрат на формализацию знаний встречаются у оболочек, в которых в окончательном виде заранее воплощены все допустимые способы представления знаний и методы дедукции, так что для создания готовой к эксплуатации интеллектуальной системы остается лишь заполнить базу знаний оболочки. Затраты на реализацию в этом случае могут быть близки к нулю, и если требования оболочки полностью соответствуют особенностям решаемой задачи, то и формализация знаний зачастую не нужна. Однако когда имеются несоответствия задачи требованиям оболочки, то затраты на формализацию знаний под жесткие, трудно раздвигаемые ограничения, свойственные практически всем оболочкам, начинают круто возрастать, требуя все бóльших и бóльших усилий разработчиков даже при незначительных расхождениях. Перефразируя известное изречение Кнута о методах сортировки, можно сказать и об оболочке, что когда она соответствует задаче, то она прекрасна, но если соответствия нет, то применение оболочки может быть просто ужасным.

Например, если некоторая оболочка обеспечивает решение задач только по методу обратной цепочки, то с ее помощью можно без особого труда разработать несложную интеллектуальную систему для задачи классификации, в том числе диагностики. Применение этой же оболочки для задачи планирования может быть вообще невозможным или требовать усилий, больших, чем разработка той же системы средствами окружения. Точно так же, если средства окружения не соответствуют потребностям решаемой задачи, то самый трудоемкий подход — разработка системы, ориентированная непосредственно на возможности того или иного языка программиро-

вания — на самом деле может оказаться самым выгодным с точки зрения затрат как на программирование, так и на формализацию знаний.

Не следует забывать еще один важный момент: рост затрат на формализацию знаний служит сигналом того, что используемое инструментальное средство плохо приспособлено для решения данной задачи. И даже если знания насильно занести в систему, задача скорее всего будет решаться крайне неэффективно и тем хуже, чем сложнее было подстроиться под требования инструментального средства.

Все это наводит на мысль, что эффективное инструментальное средство должно не только минимизировать затраты на собственно реализацию, на программирование интеллектуальной системы, но и максимально облегчать формализацию знаний людей. В настоящей главе рассматривается один возможный подход к созданию таких инструментальных средств. Он состоит в том, чтобы предоставить пользователям возможность не только описывать в базе знаний те же объекты и связи, что встречаются в предметной области, и такие же правила рассуждений, какими пользуются люди, но и строить те же методы рассуждений, способы решения задач, которые применяют для решения данной проблемы эксперты. За счет этого, с одной стороны, предельно упрощается формализация знаний, поскольку система всюду имеет дело с теми же объектами, правилами и способами рассуждений, что и люди. С другой стороны, система приобретает возможность решать задачи столь же эффективно, как эксперты, поскольку пользуется в точности теми же методами и приемами, что и они. Более того, воспроизведение в базе знаний способов мышления людей создает предпосылки к тому, чтобы, изучив способ работы человека, позднее заменить целые отрезки его метода другими, более эффективными или теоретически обоснованными подходами и тем самым улучшить работу системы по сравнению с человеком.

Однако методы, применяемые экспертами, значительно варьируют от одной задачи к другой, и эффективных способов рассуждений может быть, вообще говоря, столько же, сколько задач. Чтобы справиться с ростом количества способов логического вывода, которые пришлось бы реализовать в инструментальном средстве, в системе ЭКСПО используется подход, при котором различные способы решения задач строятся из одного и того же набора элементов, названных примитивами дедукции. Различным методам рассуждений экспертов ставятся в соответствие разные комбинации одних и тех же примитивов. Чтобы задать системе некоторый специфический способ рассуждений, во многих случаях достаточно лишь описать необходимую комбинацию уже имеющихся у нее примитивов дедукции. После того как комбинация примитивов, заданная с помощью описателей проблем и задач, поместится в базу знаний, там окажутся сосредоточены не только фактические знания эксперта (факты и правила), но и приемы его работы, способы решения задач. В результате инструментальное средство приобретает возможность улавливать знания экспертов и методы решения ими задач.

В целом в основу подхода, принятого для построения способов логического вывода в системе ЭКСПО, положен следующий принцип: методы

рассуждений, применяемые людьми, разбиваются на небольшое число примитивов дедукции, а затем благодаря комбинированию этих примитивов строятся интеллектуальные системы, работающие так же качественно и эффективно, как эксперты.

## 4.1. РАЗДЕЛЕНИЕ ПРОБЛЕМ НА ЗАДАЧИ

Когда перед экспертом ставят ту или иную проблему, существует разрыв между начальным состоянием, где находится эксперт, и поставленной целью. Решение проблемы экспертом можно рассматривать как деятельность, направленную на преодоление имеющегося разрыва, как поведение, направленное на достижение цели. Процесс решения проблемы складывается из серии движений — шагов, ведущих от начального состояния к цели. Каждый шаг означает переход в пространстве решений проблемы от одного состояния к другому. Величина шага зависит от того, какой уровень абстракции выбран для представления пространства решений. На самом высоком уровне существуют только начальное состояние, цель и единственный шаг — метод решения всей проблемы. Понизив уровень абстракции, получаем разделение этого метода на ряд более мелких шагов, на алгоритмы решения частных задач (рис. 4.1). Те, в свою очередь, можно разделить на еще более мелкие подзадачи вместе с методами их решения и т.д. Опускаясь все ниже и ниже, в конце концов дойдем до мельчайших элементов знаний — отдельных фактов и правил, а также примитивов, способных манипулировать этими элементами.

Каждому шагу в пространстве решений соответствует одна задача вместе с методом ее решения. Однако поскольку конкретное наполнение шага зависит от выбранного уровня абстракции, то и понятие задачи включает конструкции разного уровня: несколько подзадач можно объединить в одну задачу более высокого уровня, та, в свою очередь, войдет в некоторую еще большую задачу и так далее пока не будет получена задача самого верхнего уровня, представляющая собой всю проблему. Любая из этих конструкций является отдельной задачей, поскольку рассматривается как один шаг на выбранном уровне абстракции.

Метод решения задачи — это алгоритм шага в пространстве решений проблемы. Когда проблема делится на задачи, для которых алгоритмы (примитивы) дедукции известны, то разработчик интеллектуальной системы пользуется ими, не углубляясь в детали. Если возникает необходимость построить сложный метод рассуждений для некоторой задачи, то для нее уровень абстракции понижается на единицу, в ней выделяются подзадачи, и сначала под них подбираются (или программируются) примитивы дедукции, а затем они объединяются в метод решения первоначальной задачи. Если некоторая подзадача слишком большая, то уровень абстракции последовательно понижается еще несколько раз, до получения элементарных подзадач (рис. 4.2).

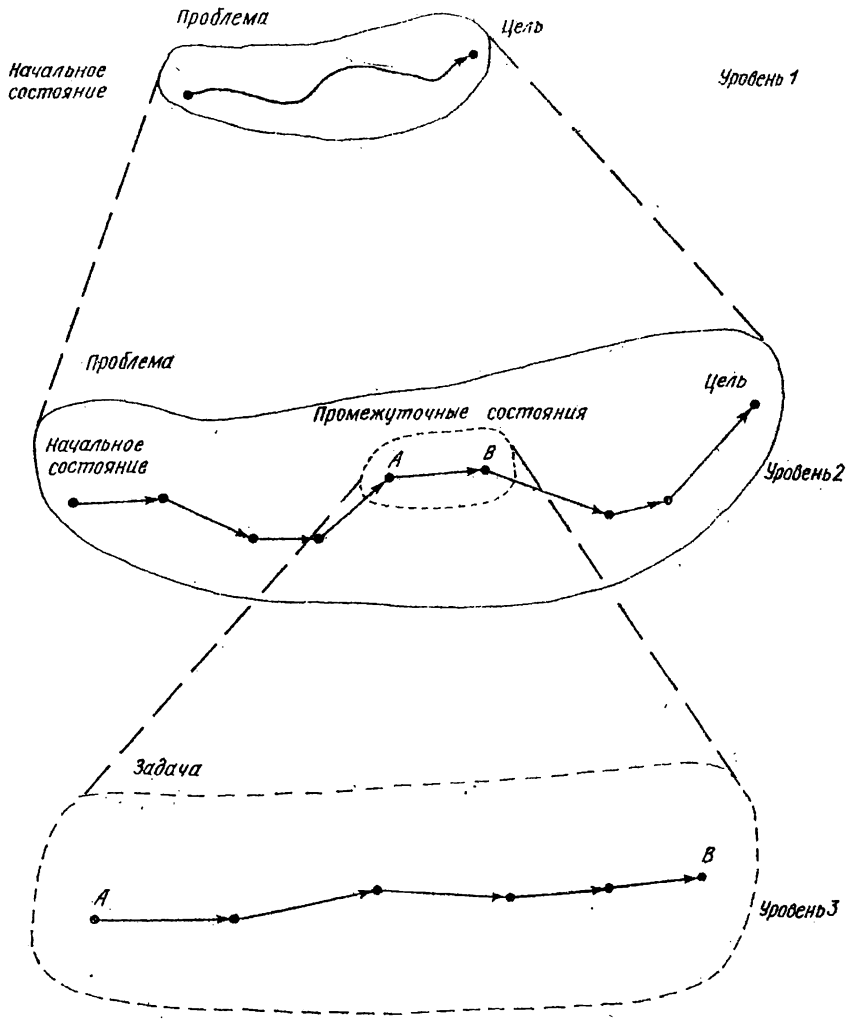



*Рис. 4.1.* Первые уровни абстракции для пространства решений некоторой проблемы (на третьем уровне показана только одна задача)

Метод решения проблемы всегда представляет собой комбинацию нескольких примитивов, решающих все те задачи, на которые делится эта проблема. Необходимая комбинация примитивов, соответствующая способу работы эксперта, строится следующим образом: сначала выясняют, на какие задачи подразделяет проблему эксперт, а затем для каждой задачи подбирают примитив, решающий ее тем же способом, что и человек. Когда какая-то из задач оказывается слишком большой и сложной, то ее дальше делят

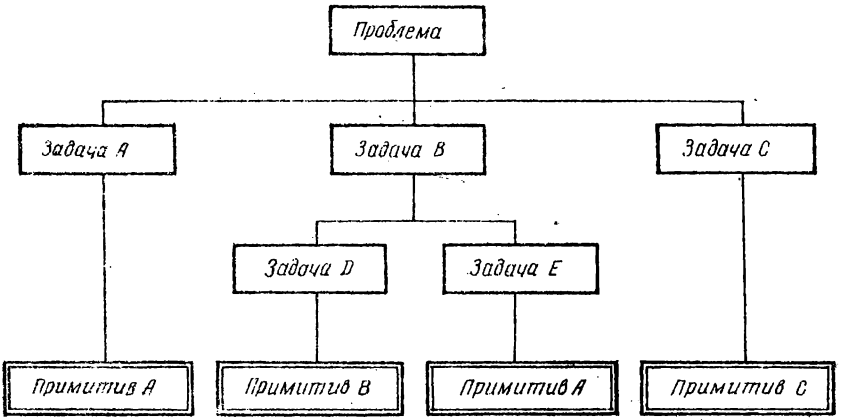



*Рис. 4.2.* Пример проблемы

на подзадачи и подбирают примитивы для каждой подзадачи отдельно, а затем комбинируют их нужным образом. Комбинирование примитивов выполняют с помощью описателей проблем и задач.

В состав инструментального комплекса программ входят примитивы, реализующие несколько общеупотребительных способов рассуждений, и во многих случаях их достаточно для построения интеллектуальной системы. Когда набор примитивов в инструментальной системе покрывает потребности разработчика, ему не приходится писать и отлаживать ни одной программы дедукции. В то же время инструментальная система открыта для расширений. В ней можно запрограммировать новые примитивы дедукции, т.е. любые новые алгоритмы решения задач, которые понадобятся разработчику интеллектуальной системы и которые он сможет описать на каком-либо выбранном им самим языке программирования. Обычно для этой цели используют Пролог.

**Описатели проблем и задач.** Согласно принятой в системе терминологии главное назначение инструментального средства — решать проблемы, описанные в базе знаний. Каждой проблеме пользователь дает название и для каждой заполняет отдельный описатель. С помощью этого описателя он задает основные этапы, шаги решения проблемы. В самом общем случае считается, что любая проблема, с которой имеет дело система, сложна и включает несколько задач. Каждая задача также имеет в базе знаний отдельный описатель.

Пример описателя для проблемы диагностики персонального компьютера приведен на рис. 4.3. Первый шаг решения этой проблемы состоит в выполнении команды печати, в результате чего пользователю выдается приглашение к диалогу с интеллектуальной системой. В данном случае предполагается, что с помощью инструментального средства создана некоторая экспертная система с условным названием ПК—ЭКСПЕРТ, от чьего имени ведется диалог. Последующие шаги решения проблемы состоят в выполнении команд, аргументами которых служат имена задач. Смысл всего описателя

```
ПРОБЛЕМА:  ДИАГНОСТИКА ПК.

Печатать:  "      З д р а в с т в у й т е.
                  Система ПК-ЭКСПЕРТ начинает работу.
                  Приступим к диагностике персонального
                  компьютера.
                  Ответьте, пожалуйста, на следующие
                  ниже вопросы".

Решить задачу:  СБОР НАЧАЛЬНЫХ СИМПТОМОВ.

Решить задачу:  ПОИСК НЕИСПРАВНОСТИ.

Печатать:       Ниже приведен перечень наиболее
                вероятных неисправностей, найденных
                мной у компьютера:
                % (неисправность компьютера)".

Решить задачу:  ВЫДАЧА РЕКОМЕНДАЦИЙ.

Печатать:  "    Мои рекомендации заключаются в
                следующем:
                % (рекомендации)".
```

*Рис. 4.3.* Описатель проблемы диагностики персонального компьютера

состоит в следующем: решение проблемы должно быть найдено путем выполнения команды печати, а затем последовательного решения трех указанных задач, сопровождающегося выдачей полученных результатов (имена печатаемых переменных указа ы в скобках после знака "%"). Строками описателя служат команды машине вывода. Это те же команды, что используются в правилах логического вывода и описаны в параграфе 3.4. Для удобства можно считать, что в описателе в наиболее общем виде задан алгоритм решения проблемы. Эту часть знаний о предметной области отличает явно выраженный процедурный характер.

Каждая задача, на которую ссылается описатель проблемы, имеет в базе знаний свой собственный описатель. К нему обращаются по имени задачи. Если какую-то из задач делят на подзадачи, то для них заводят отдельные описатели, а имена подзадач включают в основной описатель как аргумент команд "решить задачу". Действие этой команды аналогично вызову подпрограммы в языках программирования: машина вывода отыскивает в базе знаний нужный описатель задачи, последовательно сверху вниз выполняет все содержащиеся в нем команды, а затем возвращается в точку вызова, откуда было сделано обращение к задаче. Дробление задач допускается на любую глубину, какую выберет проектировщик базы знаний. Совокупность задач и связей между ними задает в базе знаний структуру проблемы.

Описатель одной задачи из трех, имеющих отношение к диагностике персонального компьютера, приведен на рис. 4.4. Эта задача решается за два ша

```
ЗАДАЧА: ПОИСК НЕИСПРАВНОСТИ.

Найти: состояние объекта ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР.
  Метод: прямой.

Найти: неисправность компьютера.
  Метод: обратный.
  Если: состояние объекта ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР
        -- неисправен .
```

*Рис. 4.4.* Описатель задачи для проблемы диагностики компьютера

га. На первом машина вывода должна отыскать значение атрибута "состояние" объекта ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ, а на втором — переменной "неисправность компьютера". Для задания цели машине вывода используют специальную команду "найти", где указывают имя искомой величины и применяемый метод логического вывода. В данном случае сначала следует отыскать значения атрибута по методу прямой цепочки рассуждений, а затем, на следующем шаге решения задачи, найти значение переменной с помощью обратной цепочки. Кроме того, во втором шаге задачи задано условие, начинающееся со слова "если". Оно означает, что логический вывод значения переменной "неисправность компьютера" нужно делать только в том случае, если компьютер действительно неисправен. Это так называемая блокировка, при которой выполнение команды обусловлено результатом анализа некоторых посторонних факторов. Такие блокировки разрешается задавать для любых команд, встречающихся в описателях проблем и задач. Условия блокировок строят в точности по тем же принципам, что и условия правил логического вывода, но смысл блокировки несколько иной — когда ее условие ложное, то текущая команда пропускается и вместо нее выполняется следующая команда из описателя.

Решить задачу всегда означает необходимость логически вывести значения одной или нескольких переменных либо атрибутов объектов. Когда речь идет о переменной, то для нахождения ее значений отыскиваются и обрабатываются все правила, где она фигурирует. Атрибуты выводятся в полном соответствии со своими описаниями, сделанными для соответствующих объектов.

Описатели проблем и задач, как правило, содержат множество команд. Например, описания задач часто начинают с нескольких команд "спросить", чтобы выяснить у пользователя особо важную начальную информацию о задаче и ускорить ее решение, отказавшись как можно раньше от ненужных ветвей рассуждений. В других случаях команды вывода значений перемежают выдачей на печать всевозможной текстовой информации, извещающей пользователя о ходе решения задачи, о переходах от одних задач к другим и т.п. Однако если отбросить все эти несущественные команды, то структура проблемы и способ ее решения задается прежде всего перечнем целей — переменных и атрибутов, значения которых следует найти, — а также спосо-

бами вывода для этих целей. Ход решения полностью определяется именами переменных, их последовательностью и используемыми методами вывода каждой величины.

Следовательно, описатель проблемы — это та часть базы знаний, где содержится обобщенный алгоритм решения проблемы. В описателе задачи задают одну или несколько целей и (возможно, разные) способы логического вывода для каждой цели. Цель представляет собой одну переменную или один атрибут объекта. Для любой цели указывают ее собственный способ вывода, которому соответствует один примитив дедукции.

**Способы описания проблем.** Описатели проблем и задач отличаются друг от друга только названием; их возможности полностью совпадают — они строятся по одним и тем же правилам, в них допускаются одни и те же команды и разрешены одинаковые блокировки. Очевидно, в этих условиях для любой достаточно сложной проблемы существует несколько вариантов разделения на задачи. Более того, разделение на задачи вообще не обязательно, и во многих случаях, проектируя базу знаний, удается обойтись единственным описателем проблемы без каких-либо задач. Отдельные задачи выделяют в тех случаях, когда это хорошо согласуется с особенностями предметной области. Например, диагностические проблемы почти всегда естественным образом разделены не менее чем на три шага: выяснение начальных симптомов, постановку диагноза и выдачу рекомендаций. Если специфика предметной области такова, что выделить относительно обособленные задачи затруднительно (обычно дело для проблем, к примеру, планирования), то можно обойтись единственным описателем проблемы, сразу задав в нем конечную цель работы системы. Однако когда проблему делят на задачи, это способствует лучшему документированию создаваемой интеллектуальной системы, а также упрощает проектирование и изменение хорошо структурированной базы знаний.

Описатели проблемы и всех входящих в нее задач в совокупности задают в базе знаний структуру решаемой проблемы. В зависимости от того, как увязаны и взаимодействуют между собой задачи, различают несколько типов проблем. К наиболее распространенным типам принадлежат линейная, иерархическая и рекурсивная проблемы (рис. 4.5). Линейная проблема делится на задачи, которые решаются строго последовательно одна за другой. Поскольку в этом случае нигде, кроме как у проблемы, нет подзадач, то отдельные описатели задач часто не заводят, обходясь единственным описателем проблемы. Этот описатель складывается из длинной последовательности команд "найти", задающих ход решения всей проблемы, задача за задачей. Иерархические проблемы имеют вид дерева, где количество ветвей, входящих в каждую вершину, равно числу подзадач в данной задаче. Здесь тоже не обязательно заводят описатели непременно для каждой задачи, вместо этого выделяют линейные участки и объединяют их в одном описателе. Именно так был построен описатель задачи поиска неисправности персонального компьютера, где в одном описателе фактически объединены две подзадачи с разными способами решения (см. рис. 4.4). Применяя этот

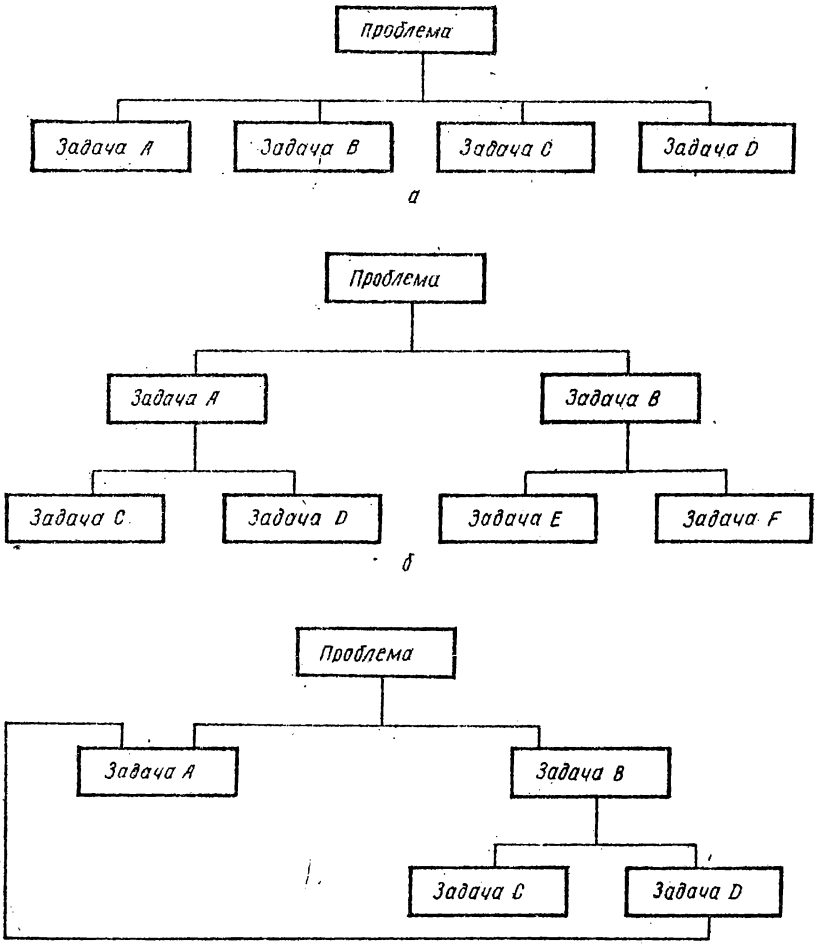



*Рис. 4.5.* Некоторые типы проблем: линейная (*а*), иерархическая (*б*), рекурсивная (*в*)

принцип и дальше, всю проблему диагностики компьютера (а именно ее структура показана на рис. 4.1) можно было бы задать в единственном описателе проблемы, включив в него четыре команды "найти" — по одной для каждой подзадачи.

Рекурсивные проблемы, вообще говоря, требуют наибольшего количества описателей, поскольку здесь делаются возвраты к ранее решенным задачам и нужно упоминать их имена. Рекурсивные цепочки строятся по схеме, проиллюстрированной на рис. 4.6. Сначала в описатель задачи включается вызов подзадачи, как это сделано для задачи А. Затем в описатель под-

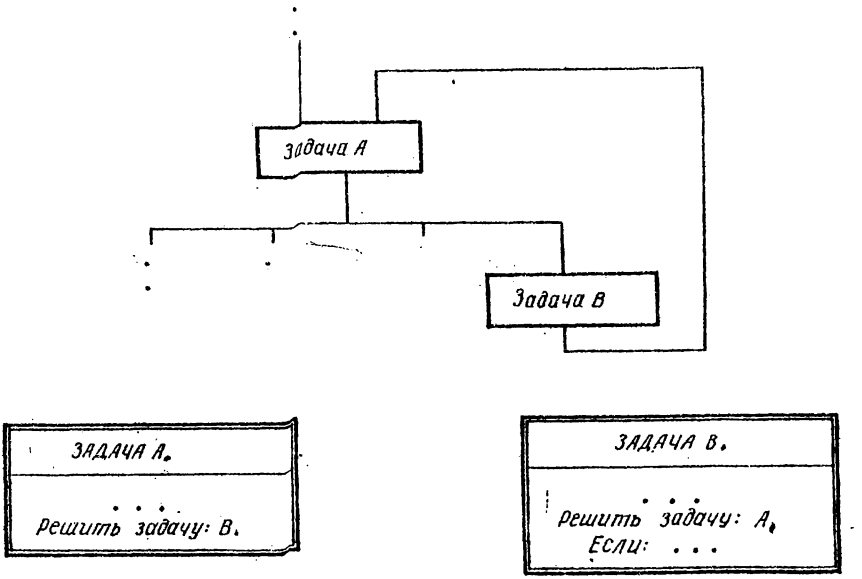



*Рис. 4.6.* Структура рекурсивной задачи и способ ее описания

задачи В включается вызов, означающий возврат к первоначальной задаче, который обычно сопровождают блокировкой. В условиях подобных блокировок часто включают хотя бы одно сравнение с "неизвестно", чтобы вернуться к первоначальной задаче, если значение необходимой величины все еще не найдено. В других случаях наборот блокировку применяют в первоначальной задаче и исключают из подзадачи. Независимо от того, какой способ используется для построения базы знаний, хотя бы одно условие прекращения рекурсии должно быть непременно задано, чтобы избежать бесконечных повторов.

В целом разделение проблем на задачи служит одновременно нескольким целям. Во-первых, оно представляет собой удобный инструмент построения сложных методов рассуждений, благодаря которому сначала проблема дробится на мелкие подзадачи, а позднее для каждой из них подбирается наилучший для нее способ решения, самый эффективный примитив дедукции. Во-вторых, оно способствует хорошей структуризации базы знаний, облегчая ее построение, документирование и последующие изменения. В-третьих, когда решение проблемы разбивается на несколько обособленных шагов, то создаются предпосылки для значительного увеличения скорости логического вывода, так как теперь машине вывода на каждом шаге уже не приходится обрабатывать непременно все правила, хранящиеся в базе знаний, а можно ограничиться лишь их подгруппой, имеющей непосредственное отношение к текущей задаче. И наконец, оно позволяет иерархически

структурировать пространство решений проблемы, чтобы эффективно управлять логическим выводом.

Последнее положение не так очевидно, как остальные, и, хотя оно тесно связано с предпоследним, нуждается в некоторых пояснениях. Иерархическое структурирование пространства решений представляет собой один из самых распространенных способов управления выводом в искусственном интеллекте. Оно используется в системе трехмерного компьютерного видения (зрения), где сначала грубо определяют внешние контуры предмета, а затем постепенно, за несколько шагов, проясняют каждую деталь изображения в отдельности, восстанавливая цепостную картину. Тот же подход встречается и в других интеллектуальных системах [MOL88]. В этом случае также применяют несколько уровней абстракции пространства решений и сначала для экономии времени обследуют это пространство в первом приближении, игнорируя детали и грубо намечая решение и область, где следует искать варианты результата. Затем уровень абстракции понижают на единицу и, анализируя первую группу деталей, еще больше уменьшают пространство поиска и количество возможных вариантов решения. Процесс продолжают до тех пор, пока не выяснены мельчайшие детали и не найден именно тот вариант результата, который подтверждается всеми имеющимися фактами.

Выигрыш от применения этого способа дедукции получается, во-первых, за счет разделения пространства решений на ряд областей и, во-вторых, из-за принятого способа перебора областей, благодаря которому, понижая уровень абстракции, машину вывода каждый раз быстро направляют в необходимую подобласть пространства решений. Для управления перебором областей и уровней абстракции применяют метазнания, т.е. "знания о знаниях". В системе ЭКСПО их роль могут играть команды с блокировками, встречающиеся в описателях проблем и задач. Условия блокировок — это тот инструмент, с помощью которого, управляя выбором задач, организуют перебор областей в пространстве решений проблемы. Однако если траектория решения сложна и запутана, то, чтобы в точности воспроизвести ход рассуждений эксперта, иногда бывает выгоднее управлять дедукцией с помощью другого инструмента — обычных правил логического вывода, в заключения которых включены команды "решить задачу". Например, показанное ниже правило запускает решение задачи, заключающейся в детальном поиске неисправности процессора, если при грубом анализе симптомов установлено, что неисправен именно этот компонент компьютера:

Если состояние объекта ПРОЦЕССОР — — неисправен,
то решить задачу ПОИСК НЕИСПРАВНОСТИ ПРОЦЕССОРА.

Это обычное правило, которое играет роль метаправила, так как управляет процессом решения задачи, организуя понижение уровня абстракции и переход к новой области пространства решений, к одной из подзадач. Подзадача, на которую ссылается правило, имеет в базе знаний отдельный описатель, причем обращение к ней производится непосредственно из правил вы-

вода и может даже нигде не встречаться в описателях проблемы или других задач.

Поскольку при обследовании каждой области пространства решений задействуются только те правила, которые в ней находятся, то иерархическая структуризация обеспечивает эффективную обработку базы знаний. Используемые для ее организации способы управления дедукцией позволяют точно задавать машине вывода сложные траектории движения от одной области пространства решений к другой, от одного уровня абстракции к другому.

**Методы управления выводом.** Система ЭКСПО допускает четыре основных способа управления логическим выводом заключений: разделение проблем на задачи и использование команд с блокировками; примененение метаправил, управляющих перебором задач; управление с помощью правил, где сравнивается переменная "текущий контекст"; применение демонов и присоединенных процедур.

Первый из них в наибольшей степени позволяет отделить собственно знания от управления выводом, так как управляющая информация сосредоточена в описателях проблем и задач. Это способствует лучшей структуризации знаний, облегчает построение базы знаний и ее последующие изменения.

Метаправила удобны для заданий особо сложных, замысловатых траекторий решения задач. И хотя в этом случае алгоритмы решения всех подзадач также хранятся в отдельных описателях, тем не менее последовательность вызова подзадач, а это тоже управляющая информация, разбрасывается по многим правилам логического вывода, и держать ее под контролем становится сложнее, чем если бы она была сосредоточена в одном месте.

Аналогичное можно сказать о наиболее простом способе управления, когда в правилах фигурирует переменная "текущий контекст". При этом подходе управляющая информация также разделяется между многими правилами, составляющими базу знаний. Хуже того, поскольку в базе знаний не выделены отдельные задачи, разделение пространства решений на области существует только логически, а не физически — здесь всегда просматриваются все правила, хотя более детально обрабатываются лишь те из них, которые соответствуют текущему контексту. По соображениям эффективности этот метод следует выбирать только для управления выводом в пределах небольших подзадач, где количество правил невелико.

Демоны и присоединенные процедуры также можно использовать для управления выводом. Например, для проблемы диагностики персонального компьютера базу знаний можно организовать следующим образом. Следует включить в описатель проблемы единственную цель — найти значение атрибута "рекомендации по починке" объекта ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ. Значением этого атрибута в описателе объекта должна служить присоединенная процедура:

Атрибут: рекомендации по починке.
Значение: решить задачу ВЫРАБОТКА РЕКОМЕНДАЦИЙ.

В ходе решения этой задачи будет обработан ряд правил, где неизбежно понадобится выяснить значения атрибута "неисправность компьютера" объекта ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ, в описание которого следует включить присоединенную процедуру ПОИСК НЕИСПРАВНОСТИ. При ее выполнении понадобятся значения нескольких атрибутов-симптомов, в чьи описания необходимо ввести присоединенную процедуру, но с именем СБОР НАЧАЛЬНЫХ СИМПТОМОВ. В результате такого проектирования будет получена база знаний, где выбор задач управляется исключительно присоединенными процедурами, а способ вывода в целом характеризуется как обратный (хотя внутри процедур могут встречаться и другие виды дедукции). Также, пользуясь демонами, нетрудно организовать вывод в прямом направлении — от симптомов к диагнозу, а затем к рекомендациям.

Демоны и процедуры как инструмент управления выводом по своим свойствам почти не отличаются от метаправил. У них тоже управляющая информация разбрасывается по многим описателям объектов (не правил), но от этого контролировать ее нисколько не легче.

Возможности, предоставляемые четырьмя способами управления выводом, во многом перекрываются, и, выбирая тот или иной подход, следует всегда руководствоваться простым принципом: в каждом конкретном случае использовать тот метод, который наиболее естествен для данной предметной области. Это упростит построение базы знаний и обеспечит наибольшую эффективность создаваемой интеллектуальной системы. Следуя такому принципу при решении сложных проблем, для точного воспроизведения способа работы эксперта часто необходимо применять все четыре метода управления одновременно, либо чередуя их друг с другом, либо вкладывая один в другой.

## 4.2. ПРИМИТИВЫ ДЕДУКЦИИ

Описатели проблем и задач, метаправила и присоединенные процедуры представляют собой средства управления выводом на, так сказать, глобальном уровне. Однако ни одна глобальная идея никогда не была реализована, если у нее не существовало поддержки в самом низу. Такую поддержку в инструментальной системе ЭКСПО обеспечивают примитивы дедукции — алгоритмы низкого уровня, работающие с фактами, правилами, объектами и всем остальным, что имеется в базе знаний. Каждый примитив реализует какой-то один способ логического вывода, необходимый в интеллектуальной системе. Он представляет собой алгоритм решения одной частной задачи или более точно — достижения одной цели, когда под целью понимаются все значения переменной или атрибута. Уместно заметить, что любое правило, имеющее N фраз условия и одно заключение, фактически одной цели ставит в соответствие N подцелей; поэтому одна цель плюс правило всегда равносильны N целям, и ограничение, что цель — это только одна переменная или атрибут, на самом деле не является сколько-нибудь существенным.

Когда проблема складывается из нескольких задач и включает множество целей, то для каждой из них можно указать ее собственный примитив дедукции, позволяющий вывести ее наилучшим способом. Комбинация примитивов, построенная с помощью описателей проблем и задач, а также метаправил и процедур, образует метод решения всей проблемы. Иногда он так же напоминает примитивы, из которых построен, как самолет гайки и болты. Множество примитивов для проблемы покрывает все типы встречающихся в ней задач и при этом любой примитив может использоваться многократно для решения нескольких различных, но однотипных задач.

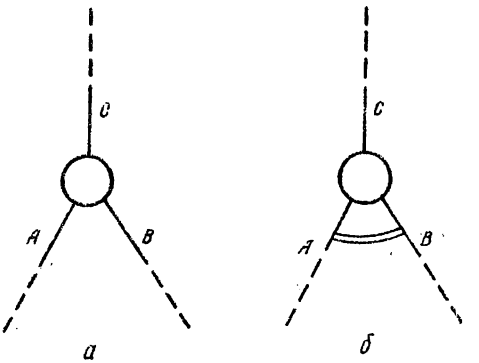



*Рис. 4.7.* Представление правил с помощью узлов и дуг: *а* — если *A* и *B*, то *C*; *б* — если *A* или *B* то *C*

Базовый набор примитивов, входящий в состав инструментальной системы, состоит из нескольких общеупотребительных алгоритмов дедукции, и прежде всего прямой и обратной цепочек рассуждений. Эти способы дедукции в том или ином виде необходимы практически в любой интеллектуальной системе.

**Прямая и обратная цепочки.** Обратная цепочка рассуждений начинается с анализа одной или нескольких гипотез о решении задачи, после чего каждая гипотеза проверяется на имеющихся фактах. Например, экспертная система диагностики персональных компьютеров выдвигает две гипотезы о любой ЭВМ — либо она неисправна, либо в полном порядке. Далее каждая гипотеза проверяется на имеющихся фактах (результатах тестов, сообщениях об ошибках программ и устройств и пр.), в результате одна из гипотез признается истинной, а другая отбрасывается как неподтвердившаяся. В формальной логике обратной цепочке рассуждений соответствует нисходящая процедура доказательства, еще называемая поиском сверху вниз, т.е. от гипотез к фактам [КОВ90].

В интеллектуальных системах цепочки рассуждений строятся из правил логического вывода. Цепочки возникают из-за того, что выводы, заключения одних правил фигурируют в условиях других, и правила оказываются логически связанными друг с другом. На рисунках эти связи изображают в виде дерева или сети, где узлы соответствуют правилам, а дуги — условиям и заключениям этих правил (рис. 4.7).

Известны несколько вариантов обратной цепс ки, отличающиеся дру. от друга последовательностью обработки правил логического вывода (рис. 4.8). Если первым всегда выбирается правило, ведущее вниз по цепочке, то такой метод рассуждений называют поиском в глубину. Когда первым

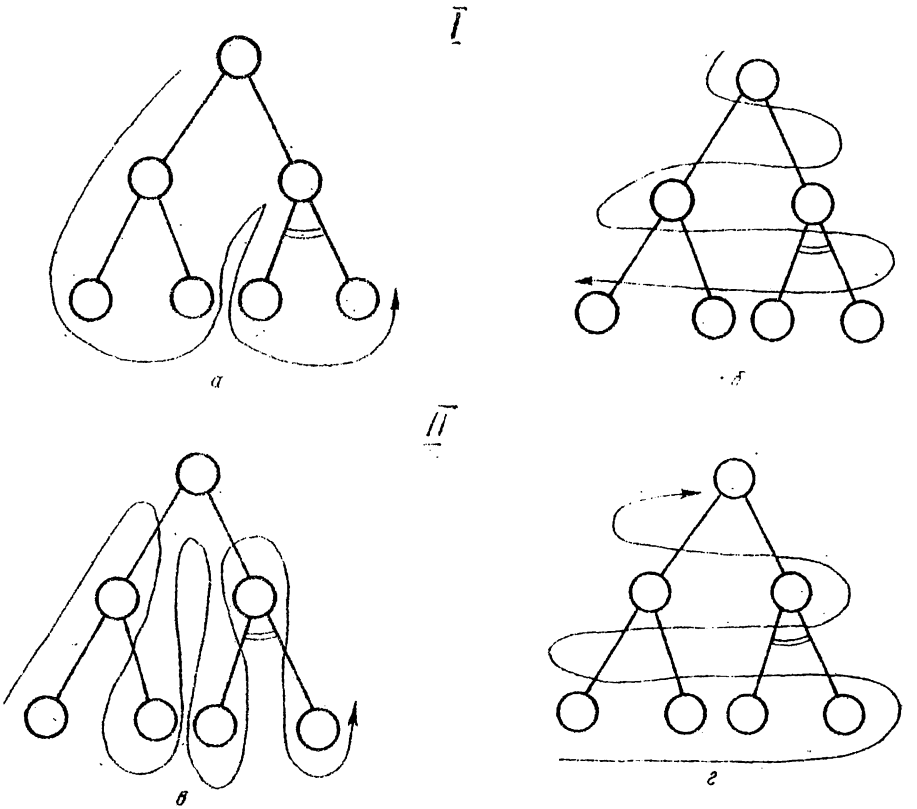



*Рис. 4.8.* Несколько общеупотребительных методов дедукции: обратная (I) и (II) прямая цепочки; *а, в* — поиск в глубину; *б, г* — поиск в ширину

всегда обрабатывается соседнее правило, находящееся на том же уровне, то в этом случае говорят о поиске в ширину.

Обратная цепочка в глубину строится следующим образом. Когда в описателе проблемы или задачи следует выполнить команду "найти", для которой указан метод обратного поиска в глубину, машина вывода отыскивает в базе знаний все те правила, где в заключениях фигурирует имя переменной или атрибута из команды. Затем она берет одно правило, выбирает самое первое сравнение из его условия и отыскивает все правила, в которых выводятся переменные или атрибуты, встречающиеся в этом сравнении. Продвижение вглубь цепочки продолжается до тех пор, пока не найдено правило, в чьем условии встречаются элементарные факты, которые либо уже известны (например, константы), либо могут быть выяснены с помощью соответствующих вопросов. Обработав условие этого правила, машина вывода поднимается на один шаг вверх и берет следующее еще не рассмотренное

сравнение из условия текущего правила, а если их нет, то следующее правило. При такой обработке дерево, построенное из правил вывода, всегда просматривается сперва вглубь, а уж затем слева направо. Обратная цепочка в ширину строится точно таким же образом, только продвижение вглубь обязательно делается лишь после того, как просмотрены все правила, находящиеся на текущем уровне. Здесь дерево правил всегда просматривается сперва в ширину, а затем на один шаг в глубину.

Каждый из этих подходов имеет преимущества и недостатки. Поиск в глубину применяется чаще, чем в ширину. Он позволяет сосредоточиться на какой-то одной линии рассуждений и методично обследовать ее на всю глубину, вплоть до мельчайших деталей, задавая пользователю вопросы для выяснения необходимых фактов. После этого текущая гипотеза либо подтверждается, либо отбрасывается и выбирается следующая линия рассуждений. Поиск в ширину во многих случаях дает возможность раньше обнаруживать ложность гипотезы, за счет того что продвижение по всем линиям рассуждений выполняется равномерно. Однако могут возникнуть трудности при общении с пользователем, так как вопросы о фактах следуют с его точки зрения бессистемно, постоянно перескакивая с одной темы на другую. Но если диалог с пользователем не нужен, то поиск в ширину дает хорошие результаты.

Другой, диаметрально противоположный способ дедукции состоит в построении прямых цепочек рассуждений. Эта цепочка начинается с анализа имеющихся фактов (например, симптомов неисправности компьютера), на основе которых с помощью правил делаются логические заключения. Вывод все новых и новых заключений, подъем по цепочкам продолжается до тех пор, пока не найдено решение проблемы, т.е. не выяснено, что именно неисправно у компьютера. В логике эту процедуру называют входящей или поиском снизу вверх — от фактов к гипотезам [КОВ90]. При этом также возможны два варианта — поиск в глубину и поиск в ширину — практически с теми же достоинствами и недостатками, но для прямой цепочки уже труднее указать, какой вариант используется чаще, так как обычно применяются разнообразные и достаточно сложные комбинации обоих подходов.

Примитив, реализующий прямую цепочку, работает следующим образом. Он просматривает все правила, относящиеся к текущей задаче, и попутно задает вопросы о неизвестных фактах. Когда условие какого-либо правила оказывается истинным, то выполняет заключение правила, обычно помещая в рабочую область базы знаний новую, только что выведенную величину. Циклы просмотра и обработки правил продолжаются до тех пор, пока не выведена величина, заданная в команде "найти" для текущей задачи (либо не установлено, что вывести эту величину по каким-то причинам невозможно).

Варианты прямой цепочки, связанные с поиском в глубину, в ширину или с какой-то сложной комбинации обоих подходов, задают с помощью переменной "текущий контекст". Для этого в описатель задачи обычно включают одну команду, устанавливающую первое значение этой переменной (рис. 4.9). Последующие ее значения присваиваются непосредственно в ходе

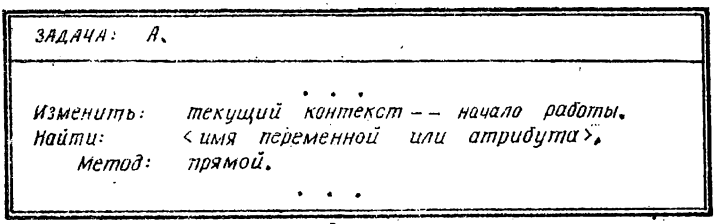



*Рис. 4.9.* Установка начального значения переменной "текущий контекст" для вывода в прямом направлении

выполнения правил логического вывода (см. параграф 3.4). Первое значение контекста для любой задачи лучше задавать с помощью команды "изменить", а не "присвоить", так как это позволяет избавиться от значений, установленных в ходе решения каких-то предыдущих задач. К слову, имя "текущий контекст" не обязательное, и вместо него можно использовать любое другое, например просто "контекст" или "текущий контекст для задачи поиска неисправностей компьютера". Однако имя для этой переменной лучше выбирать таким образом, чтобы оно отражало ее функции как инструмента управления выводом, в отличие от всех остальных переменных, служащих для накопления промежуточных и итоговых результатов.

**Приоритеты правил.** Всякий раз, когда машина вывода, отслеживая какую-либо ветвь рассуждений, пытается отыскать в базе знаний нужные ей правила, не исключена ситуация, что этих правил несколько и ей придется выбирать последовательность их обработки. Эта ситуация возникает при обратной цепочке рассуждений, когда необходимо определить последовательность просмотра правил, находящихся на одном уровне дерева или сети, т.е. всех тех правил, в которых выводится нужная в данный момент величина. При прямой цепочке рассуждений, когда в каждом цикле просматриваются все правила, относящиеся к текущей задаче, множество кандидатов на обработку еще больше и выбор последовательности продвижения машины вывода часто оказывает определяющее влияние на эффективность решения всей задачи.

Для повышения скорости решения задач при прямой и обратной цепочках имеет смысл сначала просматривать ту линию рассуждений,где находится наиболее вероятный результат либо выясняются наиболее важные факты, в максимальной степени сужающие область дальнейшего поиска. Чтобы задать нужную последовательность обработки правил, каждому из них можно присвоить приоритет — число из интервала от 0 примерно до 32000. Порядок обработки правил с учетом их приоритета выглядит следующим образом. Когда в результате поиска в базе знаний отобрано более одного правила-кандидата, то первым обрабатывается то, которое имеет наивысший приоритет. Если приоритеты правил одинаковы, то база знаний просматривается в той последовательности, в какой создавалась, т.е. первое занесенное туда правило обрабатывается первым, второе — вторым и т.д.

```
ПРАВИЛО:  П-070.

Приоритет:  1000.

Переменная:  состояние компьютера.
  Значение:  неисправен.

    Если:     состояние  объекта  МОНИТОР     -- неисправен
          или  состояние  объекта  ПРОЦЕССОР   -- неисправен
          или  состояние  объекта  КЛАВИАТУРА  -- неисправна.

    Объяснение:
      "
       ...
      Автор:  Иванов А.Б.
      Дата:  15.02.91.
      Источник:  ..."
```

*Рис. 4.10.* Правило с приоритетом 1000

Правило из гл. 3 с приоритетом 1000 приведено на рис. 4.10. Если в описателе правила фраза, в которой задан приоритет, пропущена, то по умолчанию считается, что он равен 100. Когда эта фраза пропущена во всех правилах, составляющих базу знаний, они обрабатываются в том порядке, в каком хранятся. Позднее в ходе тестирования и изменения базы знаний туда можно добавить новые правила, например с приоритетами 50 и 150, изменяющие первоначальную последовательность рассуждений.

Задание приоритетов правил — это наиболее тонкий инструмент управления логическим выводом по сравнению с описателями задач, демонами, присоединенными процедурами и метаправилами. Он как бы вкладывается внутрь каждого из этих более грубых механизмов, позволяя точнее настраиваться на нужды той или иной предметной области и влиять на эффективность решения задач. Однако, создавая базу знаний, сразу согласовать приоритеты всех помещаемых туда правил нелегко. Построение интеллектуальной системы упрощается, если подключать механизм приоритетов на последних этапах работы над базой знаний, когда она уже прошла необходимое тестирование и корректность всех линий рассуждений подтверждена. Пересмотрев после этого приоритеты правил, иногда удается значительно поднять качество диалога с пользователем и эффективность решения проблемы.

**Построение метода вывода.** Проиллюстрируем конструирование способа логического вывода на примере простой задачи, где тем не менее необходим достаточно замысловатый метод дедукции, использующий три рассмотренных выше примитива — обратную цепочку с поиском в глубину, обратную цепочку в ширину и прямую цепочку. Пример показывает, что принятый в системе ЭКСПО подход позволяет строить даже такие методы вывода, при которых буквально каждое правило обрабатывается своим собственным примитивом дедукции, а машина вывода, решая задачу, постоянно переключается с одного режима работы на другой, делая это чуть ли не после каждого правила.

По-прежнему считаем, что имеем дело с проблемой диагностики персонального компьютера, в которой выделены три задачи: сбор начальных симптомов; поиск неисправности; выработка рекомендаций по починке компьютера. Предположим, что в результате решения первой из них не только выясняются главные симптомы поломки, но и определяется, действительно ли компьютер неисправен, а также грубо намечается отказавший блок, например процессор или монитор. В ходе решения задачи поиска неисправности уточняются многочисленные добавочные симптомы, рекомендуются тесты, выясняются и анализируются их результаты и, наконец, точно определяется причина неисправности каждого устройства. После этого решается третья задача, в которой в зависимости от вида неисправности отыскиваются и предоставляются пользователю детальные инструкции по ее устранению. Задачи, из которых состоит проблема, решаются последовательно одна за другой.

Рассмотрим подробнее задачу поиска неисправности, где, как нетрудно заметить, применена иерархическая структуризация пространства решений. В пространстве выделен ряд областей, по одной на каждый компонент компьютера — процессор, монитор, клавиатуру, принтер и т.д. Любую область можно структурировать еще глубже, так как все компоненты состоят из ряда деталей (плат, микросхем и механических частей). Однако остановимся на верхнем уровне абстракции. Дальнейшую детализацию, если она необходима, можно сделать по тем же принципам.

Логику решения этой задачи описывают следующие правила:

Если состояние объекта ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР — — неисправен
и состояние объекта ПРОЦЕССОР — — неисправен,
то решить задачу ПОИСК НЕСПРАВНОСТИ ПРОЦЕССОРА
и неисправность компьютера — — неисправность объекта ПРОЦЕССОР.

Если состояние объекта ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР — — неисправен
и состояние объекта МОНИТОР — — неисправен,
то решить задачу ПОИСК НЕИСПРАВНОСТИ МОНИТОРА
и неисправность компьютера — — неисправность объекта МОНИТОР.

К моменту решения данной задачи значения атрибутов "состояние" для объекта ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР, ПРОЦЕССОР и МОНИТОР уже известны (они выясняются на шаге сбора и анализа начальных симптомов), поэтому для них никакой вывод не нужен. Для удобства считаем, что компьютер состоит из двух компонентов — процессора и монитора. Правила вывода для всех остальных его блоков строятся по тем же принципам, что и для этих двух компонентов.

Один возможный способ построения базы знаний при котором для управления выводом в иерархически структурированном пространстве реше-

```
ПРОБЛЕМА:  ДИАГНОСТИКА ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА.

Решить задачу:  СБОР НАЧАЛЬНЫХ СИМПТОМОВ.

Найти:  неисправность компьютера,
  Метод:  обратный,
  Если:  состояние объекта ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР -- неисправен.

Решить задачу:  ВЫРАБОТКА РЕКОМЕНДАЦИЙ.
```

```
ПРАВИЛО:  П-010.

Решить задачу:  ПОИСК НЕИСПРАВНОСТИ ПРОЦЕССОРА.
Присвоить:  неисправность компьютера -- неисправность объекта ПРОЦЕССОР
  Если:  состояние объекта ПРОЦЕССОР -- неисправен.
```

```
ПРАВИЛО:  П-020.

Решить задачу:  ПОИСК НЕИСПРАВНОСТИ МОНИТОРА.
Присвоить:  неисправность компьютера -- неисправность объекта МОНИТОР.
  Если:  состояние объекта МОНИТОР -- неисправен.
```

```
ЗАДАЧА:  ПОИСК НЕИСПРАВНОСТИ ПРОЦЕССОРА.

Найти:  неисправность объекта ПРОЦЕССОР.
  Метод:  прямой.
```

```
ЗАДАЧА:  ПОИСК НЕИСПРАВНОСТИ МОНИТОРА.

Найти:  неисправность объекта МОНИТОР.
  Метод:  обратный в ширину.
```

***Рис. 4.11.*** **Минимально необходимое содержание описателей для задачи поиска неисправности компьютера**

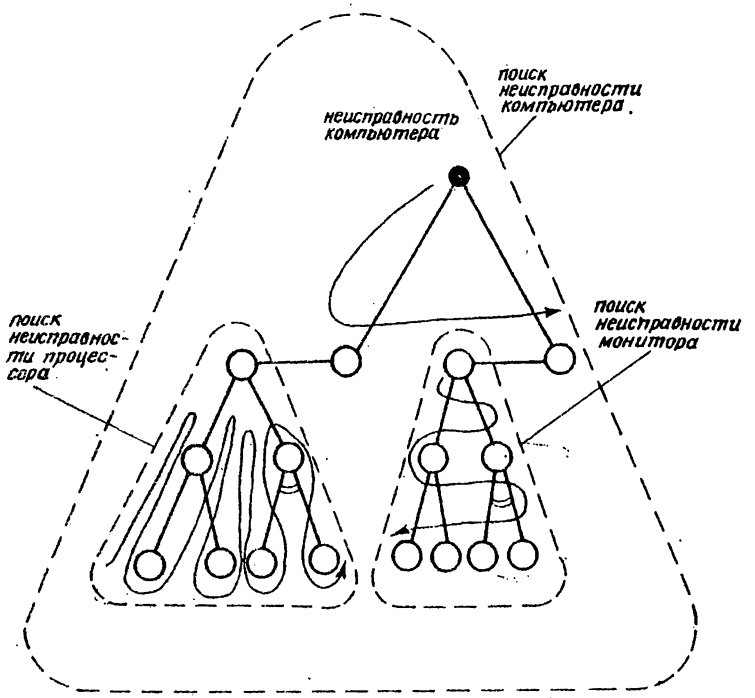



*Рис. 4.12.* Траектория вывода для задачи поиска неисправности компьютера

ний задействованы метаправила, приведен на рис. 4.11. Здесь лишь два правила обрабатываются по методу обратной цепочки рассуждений с поиском в глубину, после чего машина вывода меняет режим работы либо на прямой, либо на обратный с поиском в ширину (рис. 4.12).

В результате многоуровневого комбинирования примитивов дедукции построен способ логического вывода, соответствующий методу, обычно применяемому экспертами для решения подобных проблем диагностики. При необходимости, если задачи поиска неисправности для процессора или монитора окажутся достаточно сложными, их можно разделить на подзадачи, понизив уровень абстракции и рассмотрев их детальнее. При этом чем ниже уровень абстракции, т.е. чем мельче задача, тем проще передавать рассуждения эксперта с помощью таких простых методов, как прямая и обратная цепочки. И наоборот, чем выше уровень абстракции, а задача крупнее, тем замысловатее становится траектория решения и тем точнее улавливается специфика подхода, применяемого для той же задачи экспертом.

**Программирование примитивов.** Когда для решения некоторой подзадачи не удается подобрать примитив, точно соответствующий поведению эксперта, то разработчик интеллектуальной системы может использовать ближайший по смыслу базовый примитив или запрограммировать новый

примитив и включить в библиотеку системы, сделав его таким образом базовым. Каждый примитив, имеющий вид отдельной программы (объектного, уже оттранслированного модуля), включается в состав машины вывода создаваемой интеллектуальной системы. Компилятору базы знаний системы ЭКСПО сообщаются имя этого модуля и словесное название реализованного в нем способа дедукции. Например, трем рассмотренным примитивам соответствуют три различные программы и три названия методов — "прямой", "обратный" (по умолчанию это поиск в глубину — чаще всего используемый подход) и "обратный в ширину". Словесные названия необходимы для правильной компиляции описателей базы знаний.

Для программирования новых примитивов дедукции разработчикам интеллектуальных систем предоставлен доступ к физическим структурам базы знаний, где хранятся описатели проблем, задач, правил, объектов и вопросов, а также к структурам рабочей области базы знаний. Типичные команды, с которыми имеет дело программист, — это добавить, изменить или удалить один факт из рабочей области, прочитать или записать правило, объект, атрибут и т.д. Все физические структуры стилизованы под требования языка программирования Пролог и описаны в гл. 7. Из за богатых возможностей этого высокоуровневого языка в части описания способов логического вывода именно он рекомендуется для построения новых примитивов дедукции. В то же время каждой структуре хранения на языке Пролог соответствуют аналоги в других языках программирования. Поэтому необходимый разработчику метод вывода он может программировать на любом другом языке, какой сочтет более удобным, например на Си или Паскале, включая в свои программы команды работы с физическими структурами как внешние процедуры. Однако следует иметь в виду, что трудоемкость такого подхода к программированию методов вывода может оказаться значительно выше, чем при использовании Пролога.

Пополнение набора примитивов помогает точнее настраивать инструментальное средство на специфику той или иной предметной области и решаемой проблемы. В то же время добавление примитивов означает как бы обучение системы на примерах, позволяя ей накапливать опыт по мере решения проблем. После того как новый примитив выявлен, запрограммирован и занесен в библиотеку, всем разработчикам разрешается пользоваться им наравне с остальными способами вывода для решения проблем в других предметных областях. Одни добавляемые в инструментальную систему примитивы могут основываться на новейших достижениях когнитивной психологии и искусственного интеллекта, тогда как другие — представлять собой результаты эмпирических исследований, т.е. методы, выявленные в ходе разработки отдельных интеллектуальных систем. Независимо от того, как получены добавляемые в систему примитивы, все они равноправны, а сама система, допуская наращивание, накапливает как результаты теоретических исследований, так и практические наработки, позволяя использовать их для решения новых, все более сложных проблем.

Когда для каждой задачи удается подобрать уже имеющийся базовый примитив дедукции, то инструментальное средство применяется как оболочка интеллектуальной системы. Но если какой-то необходимый примитив в библиотеке отсутствует, то для программирования нужного способа вывода инструментальное средство используется скорее как окружение или среда, хотя комбинирование примитивов все равно осуществляется на языке высокого уровня, т.е. так, как это принято в оболочках.

Примененный в системе ЭКСПО подход к построению способов логического вывода позволил получить инструментальное средство, одновременное независимое от предметной области и способное настраиваться на особенности решаемой проблемы, причем самый сложный вид настройки — программирование метода дедукции — может понадобиться только в части, где сосредоточена специфика проблемы. Остальная часть метода логического вывода выделяется с помощью описателей проблем и задач, при этом используются алгоритмы дедукции общего назначения.

*ГЛАВА 5*

# МЕТОДЫ РАБОТЫ С НЕОПРЕДЕЛЕННОСТЬЮ

Информация, на основе которой людям приходится принимать важные решения, зачастую ненадежна, неточна, неполная и даже противоречивая. Для работы в таких условиях любая система искусственного интеллекта, если перед ней ставится задача хоть в малой степени имитировать поведение человека, должна обладать развитым механизмом дедукции, неотъемлемой частью которого являются процедуры работы с неопределенностью. Разработка процедур, способных осуществлять надежный логический вывод в условиях неопределенности знаний, выясненных у экспертов, пользователей и полученных из других источников, традиционно является одной из тех областей искусственного интеллекта, в которой активность исследований высока. Результаты этих исследований находят немедленное практическое приложение при создании разнообразных интеллектуальных систем, и, в частности, в которых применяются правила логического вывода. Даже если ограничиться лишь этим классом систем, то можно встретить около десятка теоретически обоснованных и практически проверенных, хорошо зарекомендовавших себя подходов [УЭН89, ВUX89].

В системах, основанных на применении правил вывода, знания выражены в виде конструкций "если $A$, то $C$", с помощью которых из одних утверждений (в данном случае из одного утверждения $A$) логически выводят другие (утверждение $C$). Методы работы с неопределенностью, применяемые вместе с правилами вывода, отличаются друг от друга разными подходами к ответам на пять важнейших вопросов: 1) как, с помощью чего оценивается надежность утверждений; 2) как оценивается надежность правил; 3) как, имея ненадежное утверждение $A$ и ненадежное правило "если $A$, то $C$", определять надежность утверждения $C$; 4) как следует поступать в том случае, если одно и то же утверждение, например $C$, становится известным из нескольких различных источников; 5) как оценивать надежность условия правила, когда в него входят несколько утверждений?

Среди известных способов работы с неопределенностью, проверенных в ходе эксплуатации различных интеллектуальных систем, чаще других встречаются два метода — использование коэффициентов уверенности и субъективный Байесовский подход [ФОР87, ЭЛТ87]. Оба неоднократно и успешно применялись для решения разнообразных практических задач, и каждый из них реализован не в одной интеллектуальной системе. В первом методе для

оценки надежности утверждений и правил используют неформальные, эмпирические коэффициенты уверенности, принимающие значения на шкале от −1 (когда утверждение наверняка ложно) через 0 (нечего сказать нельзя) до +1 (утверждение истинно). Коэффициенты не являются вероятностями, и для работы с ними предложены уникальные формулы, не встречающиеся в какой-либо математической теории. По мнению многих, именно слабая теоретическая обоснованность является самым серьезным недостатком этого метода [БАК92, ФОР87]. При Байесовском подходе надежность утверждений оценивают с помощью субъективных вероятностей, т.е. все-таки вероятностей, но таких, в основе которых лежит не частота появления событий, а шансы в пользу того, что утверждение окажется истинным. Комбинирование субъективных вероятностей производят по известной из теории вероятностей теореме Байеса, при этом при необходимости привлекаются формулы нечеткой логики.

Для того чтобы надежность дедукции была гарантирована, метод работы с неопределенностью должен быть теоретически обоснован. Одним из наиболее подходящих теоретических фундаментов (хотя и не единственным) служит теория вероятностей. Так, именно на ней основаны два многообещающих подхода к работе с неопределенностью, появившиеся в последние годы, — теория доказательств Демпстера — Шафера [SHA76] и теория возможностей Заде [ZAD78]. Однако при практическом применении теории ее требования, главным образом из-за высокой вычислительной сложности расчетов, обычно ослабляют. Например, при приложении подходов и Демпстера — Шафера, и Байесовского во главу угла ставится предположение о независимости свидетельств, полученных в пользу или против каждой гипотезы, тогда как любому специалисту известно, что в реальной жизни независимость свидетельств — это скорее исключение, чем правило. В этих условиях ослабление требований теории, вызванное практическими соображениями, способно поставить под сомнение надежность получаемых результатов. Для сохранения более тесной связи с теорией и реальностью метод работы с неопределенностью должен учитывать возможность того, что свидетельства окажутся зависимыми.

Другой недостаток многих подходов обусловлен тем, что неопределенность измеряют точечным значением, т.е. единственным числом — вероятностью или коэффициентом уверенности. В этих условиях, например, зная субъективную вероятность $P(A) = 0{,}7$, ничего нельзя сказать о том, что именно стоит за этим числом: то ли $P(A) = 0{,}7 \pm 0{,}2$, то ли $P(A) = 0{,}7 \pm 0{,}01$ или же $P(\sim A) = 0{,}3$. Иными словами, невозможно оценить степень надежности информации, а также отличить отсутствие уверенности от явно выраженного недоверия.

Третий и последний недостаток, на котором останавливаемся, касается неспособности многих подходов обнаруживать противоречия в знаниях. Так, когда одно и то же утверждение $A$ выведено по двум различным линиям рассуждений, причем по одной линии его вероятность равна 0,8, а по другой — 0,2, то обычно обе вероятности комбинируют каким-то образом, по-

лагая, что каждое последующее свидетельство подтверждает, усиливает первоначальное. В результате комбинирования получают вероятность, большую чем 0,8 или 0,2 в отдельности — например, оценку 0,84, вычисленную по известной из теории вероятностей формуле

$$P(A) = P(A_1) + P(A_2) - P(A_1) \cdot P(A_2) = 0{,}8 + 0{,}2 - 0{,}8 \cdot 0{,}2 = 0{,}84,$$

в соответствии с которой одно из свидетельств усиливает другое. Однако, если предположить, что обе вероятности получены в результате анализа одних и тех же данных, т.е. из зависимых источников, то в действительности имеем дело с противоречием.

Чтобы выявлять противоречия, необходимо различать зависимые и независимые свидетельства и комбинировать их так, как этого требует теория. Кроме того, необходимо раздельно накапливать свидетельства в пользу и против любой гипотезы, а также уметь сопоставлять обе оценки друг с другом. При наличии одной точечной оценки отделить свидетельства в пользу гипотезы от свидетельств против нее невозможно. Но даже и в тех вариантах метода коэффицентов уверенности, где коэффициент (КУ) вычисляется как разность меры доверия (МД) и меры недоверия (МНД) к гипотезе, КУ = МД — МНД, ситуация, когда, например, МД = МНД = 0,5 не считается противоречием. Ни в методе коэффициентов уверенности, ни при Байесовском подходе противоречивые оценки не выявляются и не обрабатываются специально, вместо этого они комбинируются, "уточняя" результат, как это делается со всеми остальными свидетельствами.

Для избавления от перечисленных недостатков удобно использовать не точечные, а иные оценки, например такие, у которых любая вероятность характеризуется некоторым интервалом, находящимся внутри отрезка [0, 1]. Поскольку речь идет о вероятности, то этот подход сохраняет тесную связь с классической теорией вероятностей и многочисленными ее ответвлениями, например с теорией доказательств. Отказ от точечных оценок позволяет без особого труда отличать неопределенность от недоверия, выявлять и исправлять возникающие противоречия в знаниях.

Идея интервальных оценок уже использовалась в ряде подходов, основанных на приложении теории Демпстера — Шафера [GAR81, BAR81], а также в экспертных системах SPERIL [ISH81] и Inferno [QUI83]. В последней системе такие оценки применялись для дедукции на сетях логического вывода. Опишем приложение этой идеи в системе ЭКСПО, использующей общеупотребительные правила вида "если А, то С". Реализованный в инструментальной системе подход основан на теории вероятностей и некоторых следствиях теории доказательств Демпстера — Шафера. Как и другие методы системы ЭКСПО, он не ориентирован на какую-то одну предметную область или один класс задач и его можно использовать для решения многих типов задач при выводе в прямом и обратном направлениях.

## 5.1. НАДЕЖНОСТЬ ФАКТОВ И ПРАВИЛ

Утверждение, что переменная или атрибут принимает некоторое значение, представляет собой один элементарный факт в базе знаний. Примеры таких утверждений: "состояние компьютера — — неисправен" или "состояние объекта ПРОЦЕССОР — — исправен". Каждый факт, т.е. каждое значение переменной либо атрибута, хранится в базе знаний вместе с двумя числами $P_{min}$ и $P_{max}$, измеряющими надежность данного факта. Любому правилу логического вывода в базе знаний также приписаны два числа $X$ и $Y$, идентификаторы которых, как смысл, иные — они не являются вероятностями, а олицетворяют силу правила. Из надежных фактов с помощью сильного правила выводится надежный результат. Если факты ненадежны или правила слабы, то ожидать надежного результата, очевидно, не приходится. Такой принцип дедукции, реализованный в инструментальной системе, хорошо согласуется со здравым смыслом. В то же время в лице теории вероятностей метод дедукции имеет математическую основу, позволяющую обеспечить необходимую надежность вывода. Для его изложения нам понадобится несколько более формальная, чем применялась до сих пор, система обозначений, которая выглядит следующим образом.

**Надежность утверждений.** Считаем, что степень неопределенности любого утверждения $A$ оценивается с помощью субъективной вероятности $P(A)$, как это принято в Байесовском подходе. Эта вероятность принимает значения от 0 до +1, на чем собственно сходство с названным подходом, использующим точечные оценки, заканчивается. Вероятность $P(A)$ задается с помощью двух величин $P_{min}(A)$ и $P_{max}(A)$, таких, что $P(A) \geqslant P_{min}(A)$ и $P(\sim A) \geqslant 1 - P_{max}(A)$. Иначе говоря, вероятность $P(A)$ лежит в интервале $[P_{min}(A), P_{max}(A)]$ (рис. 5.1). В отличие от точечной оценки характер наших

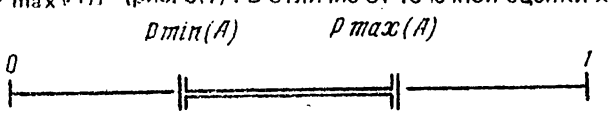



*Рис. 5.1.* Интервал для вероятности $P(A)$

знаний о надежности утверждения становится ясным: мы знаем, что точное значение вероятности неизвестно, вероятность $P(A)$ наверняка не ниже $P_{min}(A)$, вероятность $P(\sim A)$ наверняка не меньше, чем $1-P_{max}(A)$, а разность $P_{min}(A) - P_{max}(A)$ характеризует неопределенность.

При таком подходе степень доверия задается с помощью $P_{min}(A)$, недоверия — $P_{max}(A)$, а неопределенность измеряется шириной интервала. Если интервал велик, то степень неопределенности высока и мы знаем относительно немного. При $P_{min}(A) = P_{max}(A)$ знания абсолютно надежны, неопределенность отсутствует и имеем точечное значение вероятности. Однако это вовсе не означает, что сама вероятность непременно должна быть велика, отнюдь, будучи точечной, она может принимать любое значение в пределах от 0 до 1 (рис. 5.2). Если $P_{min}(A) < P_{max}(A)$, то из нескольких интервалов

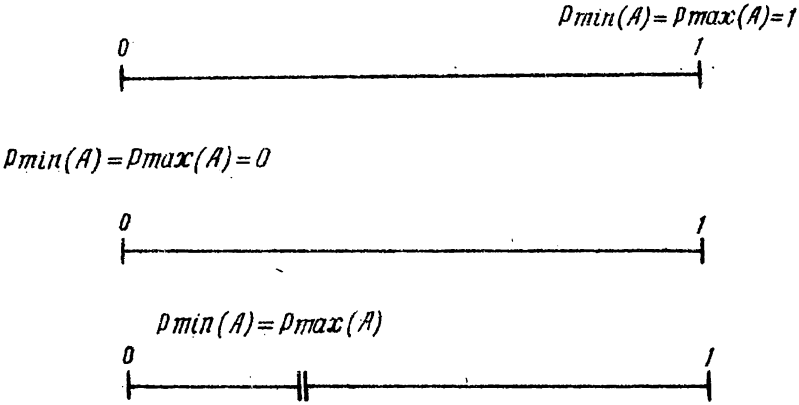



*Рис. 5.2.* Вычисление вероятности для утверждения $A$ при отсутствии неопределенности

более надежен тот, разность $P_{max}(A) - P_{min}(A)$ которого меньше (рис. 5.3). Предельный случай, когда $P_{min}(A) = 0$ и $P_{max}(A) = 1$, соответствует интервалу [0; 1] — полной неопределенности. В словесной записи это означает, что переменная или атрибут имеет значение "неизвестно".

**Сила правила.** Зададим способ измерения надежности правил, т.е. логических переходов от условий к заключениям. Пусть имеется правило "если $A$, то $C$". Обозначим условную вероятность $P(C/A)$ как $X$, а вероятность $P(\sim C/\sim A)$ как $Y$. Пользуясь формулами теории вероятностей, получим

$$\begin{array}{ll} P(C/A) & = X \\ P(A) & \geqslant P_{min}(A) \\ \hline P(C) \geqslant P(C/A) \times P(A) & \geqslant X \times P_{min}(A) \end{array}$$

$$\begin{array}{ll} P(\sim C/\sim A) & = Y \\ P(\sim A) & \geqslant 1 - P_{max}(A) \\ \hline P(\sim C) \geqslant P(\sim C/\sim A) \times P(\sim A) & \geqslant Y \times (1 - P_{max}(A)) . \end{array}$$

Отсюда

$$P_{min}(C) = X \times P_{min}(A), \qquad P_{max}(C) = 1 - Y \times (1 - P_{max}(A)).$$

Следовательно, вероятность $P(C)$ лежит в интервале $[X \times P_{min}(A), 1 - Y \times (1 - P_{max}(A))]$.

Два числа $X$ и $Y$, приписываемые каждому правилу, характеризуют силу этого правила. Если оба значения малы, то правило называют слабым. Например, при $X = 0{,}2$, $Y = 0{,}3$ и точечном, т.е. абсолютно надежном, значении $P(A) = 0{,}7$ для $P(C)$ получим интервал [0,14; 0,91] — весьма слабый результат, поскольку интервал широкий. Однако при $X = 0{,}8$, $Y = 0{,}9$ и том же зна-

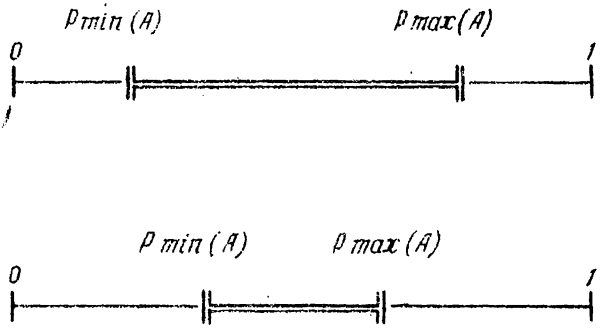



*Рис. 5.3.* Сопоставление двух интервалов надежности

чении $P(A)$ вероятность $P(C)$ уже заключена в достаточно узком интервале [0,56; 0,73]. Такое правило считают сильным, поскольку значения $X$ и $Y$ близки к единице.

Число $X$, изменяемое в интервале от 0 до 1, показывает, в какой степени истинность утверждения $A$ влияет на истинность утверждения $C$. Величина $Y$, имеющая ту же шкалу, говорит о том, в какой степени ложность $A$ позволяет судить о ложности $C$. При формулировании правил числа $X$ и $Y$ разрешается задавать независимо друг от друга, полагая, например, в одном случае $X = 0$, $Y = 1$ (истинность условия никак не дает возможности судить об истинности заключения, но ложное условие имеет большое значение для ложности заключения), а в другом — $X = 1$, $Y = 0$ (истинное условие позволяет надежно оценить левую границу интервала, а ложное условие ровным счетом ничего не говорит о его правой границе). Допускаются и различные промежуточные значения величин, расположенные между нулем и единицей, например $X = 0{,}9$, $Y = 0{,}1$ или, наоборот, $X = 0{,}1$, $Y = 0{,}9$. Такая техника задания силы правил во многих случаях не позволяет однозначно называть правила сильными или слабыми (поскольку они иногда оказываются "слабосильными" либо "сильнослабыми"), но зато дает возможность точно выразить правомерность того или иного логического заключения. Благодаря разделению силы правила на две составляющие свидетельства в пользу и против любой гипотезы накапливаются раздельно.

Силу правила указывают в его описателе отдельно для каждого заключения (рис. 5.4). Когда для какого-то заключения, как для переменной "состояние компьютера", числа $X$ и $Y$ не заданы, то по умолчанию предполагаются значения (1,1). Допускаются и другие формы сокращенной записи, например (0,7) означает (0,7; 1), тогда как (;0,7) — это (1; 0,7).

При этом не следует путать интервалы вероятностей, относящиеся только к фактам, с силой правил. Когда проектировщик базы знаний составляет описатель объекта и записывает, к примеру,

Атрибут: стоимость.
Значение: 15000 [0,9; 1], .

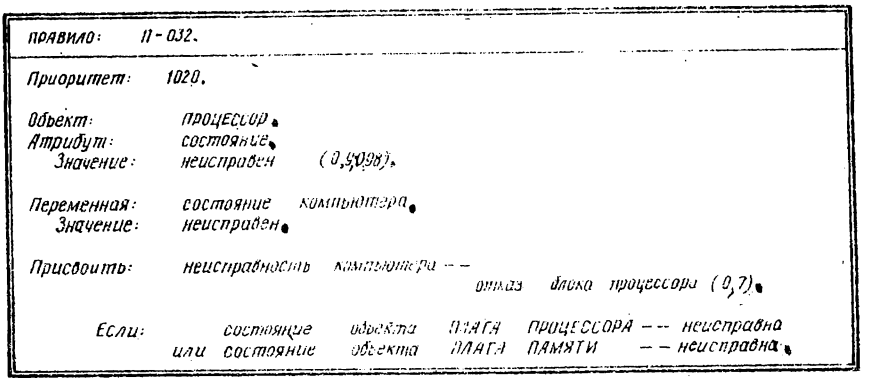



*Рис. 5.4.* Указание силы правила раздельно для каждого заключения

то он задает интервал вероятности для факта. Аналогичное делает пользователь, отвечая на вопрос системы, скажем,

≫ Укажите стоимость компьютера в рублях.
≫ 15000 [0,9; 1].

Надежность фактов записывается именно в квадратных скобках как интервал в общематематическом смысле $[P_{min}, P_{max}]$. Однако в описателе правил всегда задают силу, надежность логического перехода от условия к заключению. Сила правила, будучи просто перечнем двух чисел $(X, Y)$, заключается в круглые скобки и встречается только в описателях правил, где присутствуют необходимые логические условия и заключения.

Сила правила, заданная как $(X, Y)$, отнюдь не означает, что при абсолютной истинности условия (т.е. когда $P_{min}(A) = P_{max}(A) = 1$) заключение имеет интервал $[X, Y]$. Например, при силе правила (0,9; 0,2) и абсолютной истинности условия для заключения будет получен интервал не [0,9; 0,2], а [0,9; 1]. Сила правила определяет не интервал для заключения, а только степень расширения этого интервала по сравнению с интервалом условия. Число $X$ указывает степень понижения левой, а $Y$ — повышения правой границ интервала. При этом надежность заключения правила никогда не превышает надежности условия того же правила, что вполне соответствует нашим ожиданиям и хорошо согласуется со здравым смыслом, так как логическое заключение никогда не может быть надежнее исходных посылок.

## 5.2. КОМБИНИРОВАНИЕ СВИДЕТЕЛЬСТВ

Для сложившегося уровня технологии создания интеллектуальных систем характерно сочетание относительно несложных методов дедукции с обширными базами проблемно-ориентированных знаний, за счет объема которых в основном и компенсируется недостаточная сложность алгоритмов рассуждений. Обычной является ситуация, когда базы знаний строят таким

образом, что большинство утверждений логически выводимы по нескольким различным цепочкам рассуждений, уточняющим и перепроверяющим одна другую. Поэтому при дедукции в базах знаний часто встречается ситуация, когда одно и то же утверждение $C$ логически выводимо с помощью двух и более различных правил "если $A$, то $C_1$", "если $B$, то $C_2$" и т.д. Как скомбинировать оценки $P_{\min}(C_1)$ и $P_{\min}(C_2)$ для получения итоговой оценки $P_{\min}(C)$ и оценки $P_{\max}(C_1)$ и $P_{\max}(C_2)$ для получения $P_{\max}(C)$? Или, что то же самое, как два интервала $[P_{\min}(C_1)\,P_{\max}(C_1)]$ и $[P_{\min}(C_2), P_{\max}(C_2)]$ объединить в один $[P_{\min}(C), P_{\max}(C)]$?

**Комбинирование выводов.** Ответ на поставленные вопросы зависит от того, получены ли $C_1$ и $C_2$ из зависимых или независимых источников. Например, когда и утверждение $A$, и утверждение $B$ установлены в результате обработки двух разных цепочек правил, но в каждой цепочке на каком-то шаге тестировалась истинность одного и того же утверждения $E$, можно с уверенностью считать, что утверждения $C_1$ и $C_2$ зависимы.

Свидетельства, полученные из зависимых источников, комбинируются по простой схеме, гарантирующей нерасширение интервала вероятности:

$$P_{\min}(C) = \max_i \left\{P_{\min}(C_i)\right\}, \qquad P_{\max}(C) = \min_i \left\{P_{\max}(C_i)\right\}.$$

Для независимых источников комбинирование следует проводить совершенно иначе. Из теории известно, что вероятность события $C$ должна представлять собой сумму вероятностей независимых событий $C_1$ и $C_2$ за вычетом вероятности одновременного наступления обоих событий. Эта величина больше как $P(C_1)$, так и $P(C_2)$ в отдельности. Поэтому при выборе максимума из двух вероятностей, как это делается для зависимых свидетельств, получили бы явно заниженную левую границу интервала.

Ответ на то, как правильно комбинировать независимые свидетельства, дают теория вероятностей и теория доказательств, в частности правило Демпстера. Для случая двух свидетельств из этого правила следует [PRA85]:

$$P(C) = P(C_1) + P(C_2) - P(C_1) \times P(C_2), \qquad P(\sim C) = P(\sim C_1) \times P(\sim C_2).$$

Откуда получаем, что

$$P_{\min}(C) = P_{\min}(C_1) + P_{\min}(C_2) - P_{\min}(C_1) \times P_{\min}(C_2),$$

$$P_{\max}(C) = 1 - (1 - P_{\max}(C_1)) \times (1 - P_{\max}(C_2)).$$

Многочисленные частные случаи приложения правила Демпстера к комбинированию независимых свидетельств исследованы Гордоном и Шортлиффом при разработке расширенного метода коэффициентов уверенности [GOR85]. Ими сделано наблюдение, что в случае, когда одно из двух свидетельств, скажем $C_1$, только подтверждает гипотезу ($P_{\min}(C_1) > 0$, $P_{\max}(C_1) = 1$), а другое, $C_2$ — лишь опровергает ее ($P_{\min}(C_2) = 0$, $P_{\max}(C_2) < 1$), имеет место конфликт и первоначальная нижняя граница вероятности должна быть снижена. Для этого частного случая ими предложены формулы, в принятых нами обозначениях имеющие следующий вид ($P_{\max}(C_1) = 1$, $P_{\min}(C_2) = 0$):

```
ПРАВИЛО:    П-032.

Объект:     ПРОЦЕССОР.
Атрибут:    состояние.
Тип:        независимый.
  Значение: неисправен  (0,9; 0,98).

Переменная: состояние компьютера.
  Тип:      независимый.
  Значение: неисправен.

Присвоить:  неисправность компьютера - -
                                  отказ блока процессора (0,7).
  Тип:      независимый.

    Если:   состояние объекта ПЛАТА ПРОЦЕССОРА - - неисправна
      или:  состояние объекта ПЛАТА ПАМЯТИ      - - неисправна.
```

*Рис. 5.5.* Указание независимости свидетельств в описателе правила

$$P_{\min}(C) = \frac{P_{\min}(C_1) - P_{\min}(C_1)\times(1 - P_{\max}(C_2))}{1 - P_{\min}(C_1)\times(1 - P_{\max}(C_2))},$$

$$P_{\max}(C) = \frac{1 - P_{\max}(C_2) - P_{\min}(C_1)\times(1 - P_{\max}(C_2))}{1 - P_{\min}(C_1)\times(1 - P_{\max}(C_2))}.$$

Применение этих формул в условиях конфликта приводит к вычислению оценок, хорошо согласующихся со здравым смыслом. Так, при $P_{\min}(C_1) = 0{,}4$, $P_{\max}(C_2) = 0{,}5$ и $P_{\max}(C_1) = 1$, $P_{\min}(C_2) = 0$ вероятность $P(C)$ находится в интервале [0,25; 0,375], что точнее отражает суть конфликта, чем интервал [0,4; 1], полученный с помощью предшествующих формул. Однако при отсутствии конфликтов, т.е. во всех остальных случаях, первоначальные формулы точнее и дают легко интерпретируемые результаты: например, при комбинировании $P(C_1) = 0{,}4$ и $P(C_2) = 0{,}5$ получим интервал [0,7; 0,7] и значение вероятности останется точечным.

Для того чтобы машина вывода могла правильно обрабстать свидетельства, поступающие в пользу и против любого логического заключения, в описателе правила следует указать, какие свидетельства зависимы, а какие нет. По умолчанию предполагается, что все свидетельства, т.е. выводы правил, получены из зависимых источников. Поэтому в специальном указании нуждаются только независимые промежуточные и окончательные результаты вывода. Для этого в описатель правила помещают тип переменной или атрибута (рис. 5.5). Когда некоторая переменная или атрибут установлены константами или выводятся с помощью одного правила, то тип этой величины задавать не обязательно, поскольку независимо от него в базе знаний запоминается единственное значение вместе с интервалом его вероятности. Однако база знаний в дальнейшем может подвергаться изменениям, поэтому не исключено, что типы некоторых величин позднее придется пересмотреть. Объем работы сократится, если независимость свидетельств корректно указывать с самого начала.

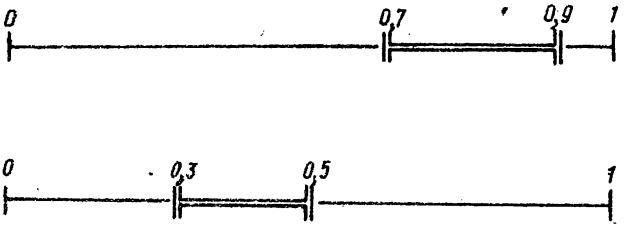



*Рис. 5.6.* Пример противоречия

**Противоречия.** Когда свидетельства получают из зависимых источников, возможно возникновение противоречий в знаниях. Пусть вероятность $P(C_1)$ находится в интервале [0,7; 0,9], а вероятность $P(C_2)$ — в интервале [0,3; 0,5]. Налицо противоречие, поскольку из одних и тех же данных логически выведены два несовпадающих интервала (рис. 5.6).

Если бы источники, на основе которых получены оба утверждения, были независимыми, то интервалы можно было комбинировать, но для зависимых источников в результате комбинирования будет вычислен интервал [0,7; 0,5]. Нарушение условия $P_{\min}(C) \leqslant P_{\max}(C)$ как раз и служит индикатором противоречия.

Противоречивые интервалы появляются по двум причинам: во-первых, в результате несогласованности введенных в базу знаний утверждений, например данных, сообщенных пользователем, и, во-вторых, вследствие некорректно заданной силы правил. Избавление от противоречий, выявленных в ходе дедукции, осуществляется следующим образом. Машина вывода обнаруживает противоречивые интервалы всегда, однако далее поступает в зависимости от того, какой задан уровень обработки противоречий. Он указывается до начала решения проблемы в специальном меню установок режимов работы машины вывода. Уровень 3 означает, что после того, как обнаружено противоречие, сообщение о нем выводится на экран дисплея в окно диалога, и решение задачи приостанавливается. Далее машина вывода выявляет все факты, влияющие на полученный результат, и просит пользователя пересмотреть относящиеся к делу ответы на вопросы. Если и после этого противоречие не устранено, то скорее всего имеется ошибка при задании надежности констант или силы правил. Ее нетрудно выявить с помощью подсистемы объяснений, задавая вопросы КАК и ЗАЧЕМ. После этого базу знаний следует изменить и начать решение проблемы сначала.

Уровень 2 обработки противоречий предполагает, что выяснение причин должно быть отложено на более позднее время. Машина вывода, обнаружив интервал, где $P_{\min}(C) > P_{\max}(C)$, выводит сообщение на экран и предлагает пользователю пересмотреть ответы на некоторые вопросы. Если противоречие устранить не удается, она устанавливает

$$P_{\min}(C) = \min_i \left\{P_{\min}(C_i)\right\}, \qquad P_{\max}(C) = \max_i \left\{P_{\max}(C_i)\right\}$$

и продолжает решение проблемы. Сообщение о противоречии попадает в протокол диалога пользователя с интеллектуальной системой и позднее может быть проанализировано, а база знаний исправлена. Уровень 1 требует от машины вывода выполнить те же самые действия, что и для второго уровня, однако никакие сообщения о противоречии ни на экран, ни в протокол диалога не выводятся и пользователя не просят пересматривать какие-либо ответы на вопросы.

**Комбинирование условий правил.** Правила, встречающиеся в реальных базах знаний, обычно имеют достаточно сложные условия, состоящие из нескольких утверждений. Например:

если $A_1$ и $A_2$ и ... и $A_n$, то $C$

либо

если $A_1$ или $A_2$ или ... или $A_n$, то $C$.

Комбинирование интервалов для подобных правил определяется тем, являются ли входящие в условие утверждения зависимыми или независимыми, а также видом логического оператора ("и", "или", "не" и т.д.).

Формулы, по которым вычисляют границы вероятностей для сложных условий, впервые выведены Квинленом [QUI83]. Их применение в контексте правил логического вывода дает возможность преобразовать каждое правило вида "если $A_1, A_2, \ldots, A_n$, то $C$" к виду "если $A$, то $C$", чтобы затем, учитывая силу правила, вычислить надежность утверждения $C$. Все формулы были выведены из очевидного неравенства, известного из теории вероятностей:

$$\max_i \{P(A_i)\} \leqslant P(A_1 \vee A_2 \vee \ldots \vee A_n) \leqslant \sum_i P(A_i).$$

Ниже приведены несколько таких формул, представляющих первоочередной интерес для баз знаний, составленных из правил логического вывода.

Независимые условия "или":

$$P_{\min}(A) = 1 - \prod_i (1 - P_{\min}(A_i)), \qquad P_{\max}(A) = 1 - \prod_i (1 - P_{\max}(A_i)).$$

Зависимые условия ИЛИ:

$$P_{\min}(A) = \max_i \{P_{\min}(A_i)\}, \qquad P_{\max}(A) = \sum_i P_{\max}(A_i).$$

Независимые условия "и":

$$P_{\min}(A) = \prod_i P_{\min}(A_i), \qquad P_{\max}(A) = \prod_i P_{\max}(A_i).$$

Зависимые условия И:

$$P_{\min}(A) = 1 - \sum_i (1 - P_{\min}(A_i)), \qquad P_{\max}(A) = \min_i \{P_{\max}(A_i)\}.$$

Отрицание "не" ("$A$ есть не $B$"):

$$P_{\min}(A) = 1 - P_{\max}(B), \qquad P_{\max}(A) = 1 - P_{\min}(B).$$

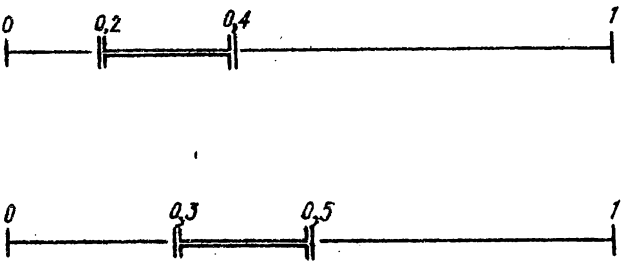



*Рис. 5.7.* Интервалы вероятностей для значений величины $A$

Зависимость либо независимость условий правила задают в его описателе, выбирая нужный вид логического оператора: частицы "и", "или" всегда соединяют независимые, а И, ИЛИ — зависимые условия. Например:

Если: состояние объекта ПРОЦЕССОР — — неисправен
ИЛИ (состояние объекта ПЛАТА ПРОЦЕССОРА — — неисправна
или
состояние объекта ПЛАТА ПАМЯТИ — — неисправна).

Логический оператор, записанный прописными буквами, означает зависимость условий, а строчными — соединяет независимые сравнения.

**Условия со сложными сравнениями.** Когда речь идет о любом утверждении $A$, входящем в условие правила, необходимо учитывать, что оно может быть непростым сравнением. Простое сравнение имеет вид $A = a$, где $a$ — константа. Например, $A = 15000$. Более сложные сравнения формулируются с помощью знаков неравенств, например $A > 15000$ или $A \leqslant 15000$.

Пусть случайная величина $A$ принимает значения $a_1, a_2, \ldots, a_n$. Каждому из этих значений соответствует свой интервал вероятности $[P_{\min}(a_i), P_{\max}(a_i)]$. Для дискретной случайной величины из теории вероятностей известно, что $P(A < a) = \sum_i P(A = a_i)$, где все $a_i$ меньше константы $a$. Отсюда выводим, что независимо от того, какая операция сравнения входит в утверждение $A$, границы интервалов вычисляются по одним и тем же формулам:

$$P_{\min}(A) = \sum_i P_{\min}(a_i), \qquad P_{\max}(A) = \sum_i P_{\max}(a_i),$$

где все $a_i$ должны удовлетворять требуемому условию, будь то $A < a$, $A \leqslant a$, $A > a$ или $A \geqslant a$.

Например, известно, что некоторая величина $A$ принимает два числовых значения $A = 36$ и $A = 38$, причем $0{,}2 \leqslant P(A = 36) \leqslant 0{,}4$; $0{,}3 \leqslant P(A = 38) \leqslant 0{,}5$ (рис. 5.7). Тогда по приведенным выше формулам для сравнения вида $A \geqslant 30$ получим интервал [0,5; 0,9], для $A > 36$ — интервал [0,3; 0,5], для $A \leqslant 40$ — интервал [0,5; 0,9] и для $A < 36$ — интервал [0; 0].

**Пример дедукции.** Проиллюстрируем дедукцию с помощью описанного метода на простом примере. Пусть имеются независимые утверждения $A$, $B$, $C$, $D$, $E$ и $F$ (рис. 5.8), логически связанные друг с другом посредством четырех правил: 1) если $A$ или $B$, то $D(X = 1; Y = 1)$; 2) если $B$ и $C$, то $E$ (0,5;

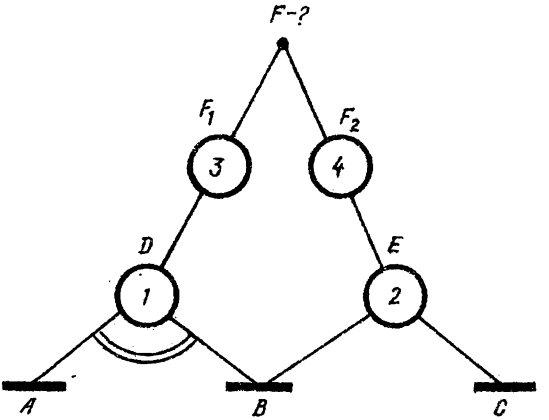



*Рис. 5.8.* Пример базы знаний для расчета надежности гипотезы

0,6); 3) если $D$, то $F$ (1; 0,7); 4) если не $E$, то $F$ (1; 0,9). Правила означают, что при известной истинности утверждений $A$ и $B$ можно судить об истинности утверждения $D$, при известной истинности $B$ и $C$ — об истинности $E$ и т.д. Истинность утверждений $A$, $B$ и $C$ следует выяснить у пользователя, задав ему соответствующие вопросы. Требуется определить, в какой мере истинна гипотеза $F$ (рис. 5.9).

Предположим, что в ходе диалога с пользователем установлены следующие интервалы для утверждений: $A$ [0; 1], $B$ [0,7; 0,9] и C (0,8; 0,9]. На вопрос об истинности утверждения $A$ был дан ответ "неизвестно", что и соответствует интервалу [0; 1].

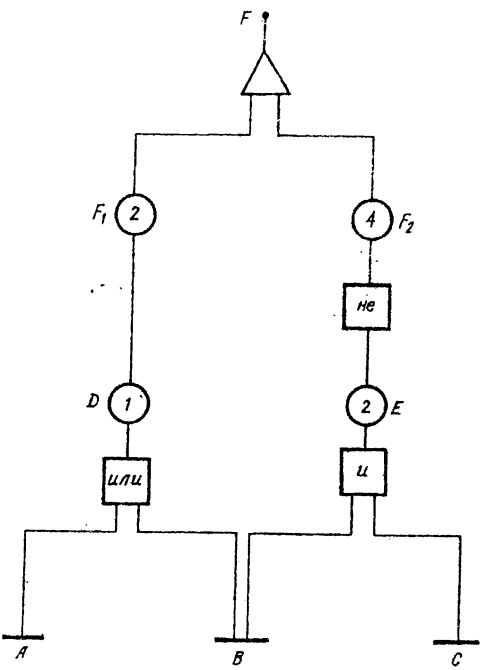



*Рис. 5.9.* Схема комбинирования условий и выводов правил примера

Первое правило имеет сложное условие, два независимых утверждения которого соединены союзом "или". Для такого правила $P_{min}(D) = Xx(1 - (1 - P_{min}(A))\times(1 - P_{min}(B))) = 1\times(1 - (1 - 0)\times(1 - 0,7)) = 0,7$, $P_{max}(D) = 1 -$

$- Y \times [1 - 1 + (1 - P_{max}(A)) \times (1 - P_{max}(B))] = 1 - 1 \times [0 \times 0{,}1] = 1$. Итак, для утверждения $D$ получен интервал $[0{,}7;\ 1]$.

Во второе правило входит союз "и", поэтому $P_{min}(E) = X \times P_{min}(B) \times P_{min}(C) = 0{,}5 \times 0{,}7 \times 0{,}8 = 0{,}28$, $P_{max}(E) = 1 - Y \times [1 - P_{max}(B) \times P_{max}(C)] = 1 - 0{,}6 \times [1 - 0{,}9 \times 0{,}9] = 0{,}886$. Для утверждения $E$ имеем интервал $[0{,}28;\ 0{,}886]$.

С помощью третьего правила предоставляется возможность получения оценки для одного из свидетельств, касающихся гипотезы $F$. Обозначим это свидетельство как $F_1$. Тогда $P_{min}(F_1) = X \times P_{min}(D) = 1 \times 0{,}7 = 0{,}7$, $P_{max}(F_1) = 1 - Y \times (1 - P_{max}(D)) = 1 - 0{,}7 \times (1 - 1) = 1$. Для свидетельства $F_1$ интервал имеет вид $[0{,}7;\ 1]$.

Четвертое правило содержит отрицание "не" и касается второго свидетельства о гипотезе. Назовем это свидетельство $F_2$ и вычислим $P_{min}(F_2) = X \times (1 - P_{max}(E)) = 1 \times (1 - 0{,}886) = 0{,}114$, $P_{max}(F_2) = 1 - Y \times (1 - 1 + P_{min}(E)) = 1 - 0{,}9 \times 0{,}28 = 0{,}748$. Для $F_2$ интервал выглядит как $[0{,}114;\ 0{,}748]$.

Поскольку оба свидетельства о гипотезе получены с использованием одного и того же утверждения $B$, то они зависимые. Комбинирование зависимых свидетельств дает $P_{min}(F) = \max \{P_{min}(F_1), P_{min}(F_2)\} = \max \{0{,}7;\ 0{,}114\} = 0{,}7$, $P_{max}(F) = \min \{P_{max}(F_1), P_{max}(F_2)\} = \min \{1;\ 0{,}748\} = 0{,}748$. В итоге интервал для гипотезы $F$ приобретает вид $[0{,}7;\ 0{,}748]$, что означает $P(F) = 0{,}724 \pm 0{,}024$, $P(\sim F) \geqslant 0{,}252$.

Интервал вероятности достаточно узкий, т.е. нами получен сильный результат, несмотря на то, что одно из правил слабое, а для сильного правила был дан ответ "неизвестно". Однако не следует сильный, надежный результат путать с высоким значением вероятности. В данном случае вероятность порядка 0,7 не очень высока, и решение может оказаться по каким-то причинам неприемлемым. Но здесь мы выходим за рамки вопросов надежности дедукции и вторгаемся в область приемлемости конкретного значения вероятности для нужд той или иной предметной области.

В целом использование интервальных оценок вместо точечных повышает информативность результатов дедукции, так как связанный с каждой гипотезой интервал ясно показывает, насколько гипотеза истинна, а насколько ложна, в какой степени знания о ней неопределенны. Поскольку для оценки неопределенности служат вероятности, то сохраняется тесная связь с теорией. Применение интервальных оценок позволяет выявлять и исправлять противоречия в знаниях. Еще одно выгодное отличие метода от многих известных подходов заключается в том, что он допускает комбинирование оценок, полученных из зависимых и независимых источников, причем вычислительная сложность работы с зависимыми свидетельствами невелика и не превышает сложности комбинирования независимых свидетельств.

*ГЛАВА 6*

# СОПРЯЖЕНИЕ БАЗ ЗНАНИЙ С БАЗАМИ ДАННЫХ

Описатели объектов, правил, проблем, задач и вопросов позволяют строить достаточно большие и сложные базы знаний, занимающие не один мегабайт внешней памяти компьютера. Фактически объем любой базы знаний может достигать такого размера, какой только поместится на внешние накопители используемой ЭВМ. Что касается сложности помещаемых в систему знаний, то она определяется интенсивностью применения таких средств, как наследование и заимствование значений, присоединенные процедуры и демоны, сложные стратегии логического вывода, а также механизмы полномасштабной обработки неопределенности и противоречий. Вместе с тем существует простой путь еще большего расширения возможностей интеллектуальных систем, строящихся с помощью инструментального комплекса программ ЭКСПО. Речь идет о сопряжении баз знаний с базами данных, которое, с одной стороны, позволяет переложить на базу данных функции хранения и выборки большого количества фактической информации (главным образом многочисленных констант), а с другой — открывает очевидный путь объединения интеллектуальных и любых других компьютерных систем, организуя обмен информацией между ними через общую базу данных.

Объединение баз знаний и баз данных предполагает не только неограниченный обмен между ними, но и согласование форм представления тех и других. Однако способы, применяемые для представления знаний и данных, существенно отличаются. В области баз данных преобладающие позиции занимает реляционная модель, которая считается наилучшей основой для эффективного решения всего комплекса проблем, связанных с хранением и выборкой данных [COD82]. В области баз знаний приоритет принадлежит двум иным формам — правилам логического вывода и фреймам, которым в системе ЭКСПО соответствуют описатели правил и объектов.

При согласовании столь разнородных способов организации информации выявлен ряд принципиальных различий между сложными формами

представления знаний и относительно простыми реляционными структурами, делающими невозможным отображение одних в другие [LAF82]. Так, фреймы в отличие от реляционных таблиц предназначены не только для хранения цифровых или символьных констант, но и для описания правил, структур управления логическим выводом, процедур обработки данных и т.д. В этих условиях один из возможных способов согласования фреймов (описаний объектов) с таблицами состоит в том, чтобы выделить те возможности, с которыми справится реляционная модель, и переложить их на систему управления базой данных, оставив за базой знаний остальные функции, которые невозможно или нецелесообразно возлагать на базу данных. Поскольку база знаний не сливается с базой данных, речь идет не о полном объединении, а скорее о шаге в этом направлении — о подключении базы знаний там, где это оправдано.

Такой подход уже применялся для сопряжения фреймовых баз знаний систем ART и KEE с внешними базами данных [MOA87]. К примеру, пакет KEE Connection обеспечивает перевод команд системы KEE в запросы к реляционной базе данных и обслуживает передачу данных между фреймами базы знаний и таблицами [ABA87]. В качестве интерфейса взаимодействия с базой данных используется язык SQL, считающийся стандартным при работе с реляционной моделью. Опишем приложение этих же принципов к структурам баз знаний системы ЭКСПО, ряд возможностей которой в точности соответствуют принципам, обычным для фреймовых систем. Примененный в ней подход позволил примерно на порядок сократить количество реляционных таблиц, используемых для хранения знаний. За счет этого снижено количество соединений при выборке данных — одной из самых сложных и продолжительных реляционных операций. Упрощена структура базы данных, и для хранения, к примеру, 50 фреймов классов понадобится порядка 50 таблиц, а не несколько сотен, как в [ABA87], что выходит за границы возможностей проектировщика.

## 6.1. ВОПРОСЫ К БАЗЕ Д НН ЫХ

Наиболее просто базы знаний и базы данных соединяются при возникновении необходимости выборки отдельных констант (значений переменных или атрибутов) из существующей базы данных. Связь реализуется с помощью описателей вопросов несколько иного содержания, чем те, которые используются при организации обычного диалога с пользователем, когда текст вопроса выдается на экран дисплея, а ответ читается с клавиатуры. Новые описатели содержат словесный идентификатор внешней программы, выполняющий доступ к базе данных, а также параметры, передаваемые от системы к программе и обратно. Для доступа к внешней базе данных можно

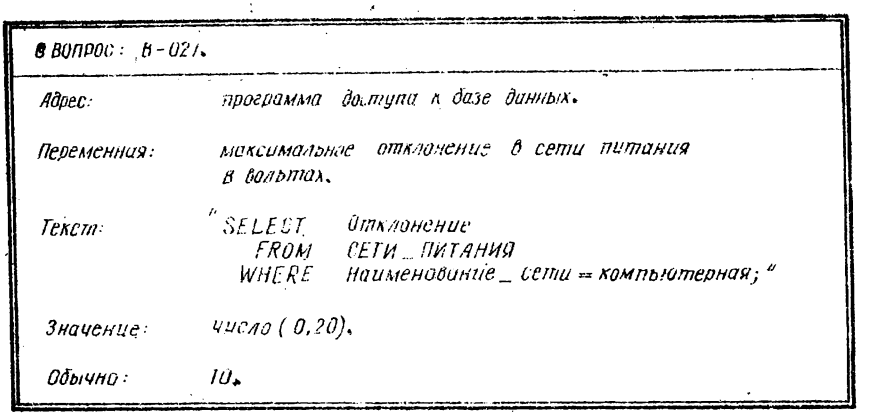



*Рис. 6.1.* Описатель вопроса, предназначенный для обмена данными с внешней программой

использовать любую СУБД, с помощью которой эта база данных создавалась и которая удовлетворяет требованиям разработчика интеллектуальной системы. Роль СУБД может играть и любая другая программа хранения и выборки данных, даже созданная пользователем самостоятельно на одном из языков программирования.

**Выборка фактов.** Пусть при решении задачи диагностики персонального компьютера необходимо выяснить максимально возможное напряжение в сети электрического питания. Эти данные в базе знаний отсутствуют, но их можно получить из базы данных об организации или учреждении, где установлен тестируемый компьютер. Для чтения необходимых сведений из базы данных составляют описатель вопроса, где указывают, что нужно найти, как искать и что делать с результатом дальше (рис. 6.1). Описатель такого вопроса начинается со служебного символа *"С"*, означающего, что вопрос не должен задаваться пользователю, а вместо этого его следует передавать для исполнения внешней программе. Имя программы указывают в описателе вопроса как адрес, куда передается запрос и откуда присылается ответ. В качестве текста вопроса в данном случае использована команда на реляционном языке SQL, требующая от СУБД отыскать величину отклонения в сети питания при условии, что сеть, представляющая интерес, — компьютерная, а не осветительная или какая-то иная. Применение языка SQL подразумевает, что для доступа к фактам используется реляционная СУБД, понимающая и способная выполнить эту команду. При необходимости вместо этой команды можно поставить любой другой текст вопроса, однако он должен быть понятен исполняющей его программе.

Машина вывода, обрабатывая такой запрос, передает его не диалоговому

компоненту инструментальной системы, а внешней программе пользователя. Точное имя программы, например PROG1, задается компилятору баз знаний в специальной таблице, где указываются ее словесный идентификатор (в данном случае "программа доступа к внешней базе данных") и имя объектного, оттранслированного модуля. Программа должна быть заранее написана на одном из языков программирования, скомпилирована и помещена в библиотеку инструментальной системы. Когда запрос на выборку данных следует передать для исполнения какой-либо СУБД, то необходимая программа проста — она содержит лишь обращение к СУБД, оформленное по принятым для этой системы правилам, а также команды чтения ответа от СУБД и его передачи инструментальной системе. Программа имеет вид внешней процедуры; первый параметр которой содержит текст запроса, а второй — список ответов, найденных в базе данных. Здесь же указывают различные детали взаимодействия СУБД с интеллектуальной системой. Например, в одном случае модуль СУБД, осуществляющий доступ к данным, загружают в память ЭВМ один раз и многократно используют для обработки запросов, в другом — его загружают при каждом обращении за данными и немедленно высвобождают память после окончания обработки запроса.

Благодаря тому что система ЭКСПО имеет доступ к внешним базам данных, хотя и оставляет за СУБД фактическое исполнение запросов на выборку фактов, с ее помощью можно строить системы нескольких типов, не все из которых являются экспертными. Так, допускается ее использование как замкнутого диалогового комплекса программ, расширяющего возможности обычных СУБД в сторону их интеллектуализации. С ее помощью организуют более сложную выборку данных, чем выполняет обычная СУБД, к примеру формулируя такие запросы, в соответствии с которыми данные не просто включаются или не включаются в результат, а включаются вместе с интервалом их надежности либо включаются только в том случае, когда интервал попадает в некоторый диапазон приемлемости. Для сложных условий выборки можно организовать комбинирование интервалов надежности, а затем их контроль на удовлетворение некоторым наперед заданным границам. В более сложных случаях найденные СУБД факты используют как исходные данные для правил дедукции, с помощью которых логически выводят нужный пользователю результат. В данном случае речь идет не только о выборке констант, но и о логическом выводе на основе фактов, полученных из базы данных. Во всех этих подходах для реализации обмена сообщениями с пользователем разрешается применять стандартный диалоговый компонент инструментальной системы или использовать вместо него программу пользователя, подменяющую во время своей работы стандартные средства.

Кроме того, комплекс ЭКСПО, используемый совместно с некоторой СУБД, может служить частью еще большей системы, где он является компонентом, ответственным за доступ к базе знаний, а следовательно, и данных. Запросы, относящиеся к выборке только фактов, компонент просто пропускает через себя и передает для исполнения подключенной к нему СУБД.

За собой он оставляет выполнение всех операций, связанных с интеллектуализацией доступа к данным и логическим выводом. Такой режим функционирования комплекса программ уже больше напоминает не интеллектуальную систему, а систему управления базой знаний.

**Использование внешних подпрограмм.** Пользуясь описателями вопросов, нетрудно организовать обмен данными не только с СУБД, но и любой другой внешней программой, реализующей какие-либо дополнительные функции, которые могут понадобиться в интеллектуальной системе. Например, с помощью внешней подпрограммы организуют общение с пользователем более сложное по форме, чем допускает стандартный диалоговый компонент. В ходе такого диалога иногда подключают средства обработки графической информации или используют какие-либо нестандартные устройства обмена с пользователем. Кроме того, внешние программы удобны для получения и первичной обработки информации, поступающей непосредственно от датчиков или электронных устройств, как это необходимо в системах управления технологическими процессами и других системах реального времени. Во всех этих случаях выход интеллектуальной системы в окружающий мир берет на себя программа пользователя, заменяя во время своей работы стандартный диалоговый компонент.

Процедуры пользователя представляют собой обычные программы, написанные на любом удобном языке. Пожалуй, единственное, предъявляемое к ним требование — по возможности они должны занимать не много памяти и не сильно замедлять работу машины вывода. Результаты своей деятельности, например первичной обработки сигналов от датчика с помощью быстрых преобразований Фурье, они должны возвращать интеллектуальной системе одним из параметров внешней процедуры.

Поскольку инструментальный комплекс открыт для перепрограммирования любого своего компонента, с его помощью допускается построение различных интеллектуальных или обладающих лишь элементами искусственного интеллекта систем, где и поступление информации, и логический вывод, и обработка результатов вывода могут проводиться программами пользователя, работающими в обход стандартных средств. Допускается также построение систем, в которых часть работы берут на себя стандартные компоненты, а оставшуюся долю выполняют программы пользователя, и в этом случае говорят о частичном использовании стандартных средств. Наконец, во многих случаях программы пользователя попросту не нужны, если все необходимые функции не содержат чего-то необычного и их в состоянии выполнить обычные компоненты инструментальной системы.

## 6.2. РЕЛЯЦИОННЫЙ ПОДХОД К УПРАВЛЕНИЮ БАЗАМИ ДАННЫХ

Общепринятым лидером в области построения систем управления всевозможными базами данных является реляционный подход [ДЕЙ90]. Он обеспечивает лучшее на сегодня решение всех проблем, связанных с управ-

| Наименование | Совместимость | Изготовитель | Стоимость |
|---|---|---|---|
| НЕЙРОН | IBM PC/XT | ПО им. Королева, г. Киев | 15000 |
| ЭЛЕКТРОНИКА | IBM PC/XT | 'Процессор', г. Воронеж | 25000 |

ТИПЫ_ОС

| Наименование | Тип_ОС |
|---|---|
| НЕЙРОН | НЕЙРОН-ДОС |
| НЕЙРОН | MS/DOS |
| НЕЙРОН | PC/DOS |
| ЭЛЕКТРОНИКА | MS/DOS |
| ЭЛЕКТРОНИКА | PC/DOS |

*Рис. 6.2.* Данные о персональных компьютерах

пением гигантскими базами данных: создания распределенных баз данных, разработки тесно интегрированных семейств, включающих средства проектирования логических и физических баз данных, генерации приложений, обслуживания справочников или каталогов данных; его применение расширяет диапазон и возможности СУБД в части управления транзакциями, мультидоступом и восстановлением [COD82]. Немаловажно также, что он предлагает наилучшую на сегодняшний день теоретическую и практическую основу для совершенствования всего комплекса информационных технологий, связанного с обработкой данных, и в частности, созданием дедуктивных баз данных как непременного компонента будущих интеллектуальных систем. Неслучайно именно к нему пристально присматриваются разработчики ЭВМ пятого поколения, проектируя средства хранения данных для создаваемых ими компьютеров.

**Реляционная модель данных.** Реляционный подход к управлению базами данных и соответствующая модель данных были предложены Коддом в 1972 г. Понятие модели данных по Кодду включает: формы представления данных и язык манипулирования принятыми формами представлений. В качестве формы представления в реляционной модели выбраны обычные таблицы, состоящие из строк и столбцов. Каждая строка таблицы описывает один объект предметной области, а имена столбцов соответствуют названиям атрибутов этих объектов. Названия таблиц обычно совпадают с названиями классов, куда входят объекты.

На рис. 6.2. показаны данные о персональных компьютерах в реляционной форме представления, а также приведены две таблицы с именами ПК и ТИПЫ_ОС. Каждый объект (НЕЙРОН, ЭЛЕКТРОНИКА) имеет атрибуты "наименование", "совместимость", "тип_ОС" и т.д. Для краткости в обеих

| Наименование | Совместимость | Изготовитель | Стоимость |
|---|---|---|---|
| НЕЙРОН | IBM PC/XT | ПО им. Королева , г. Киев | 15000 |

*а*

| Наименование | Стоимость |
|---|---|
| НЕЙРОН | 15000 |
| ЭЛЕКТРОНИКА | 25000 |

*б*

| Наименование | Совместимость | Изготовитель | Стоимость | Тип_ОС |
|---|---|---|---|---|
| НЕЙРОН | IBM PC/XT | ПО им. Королева, г. Киев | 15000 | НЕЙРОН-ДОС |
| НЕЙРОН | IBM PC/XT | ПО им. Королева, г. Киев | 15000 | MS/DOS |
| НЕЙРОН | IBM PC/XT | ПО им. Королева, г. Киев | 15000 | PC/DOS |
| ЭЛЕКТРОНИКА | IBM PC/XT | 'Процессор' , г. Воронеж | 25000 | MS/DOS |
| ЭЛЕКТРОНИКА | IBM PC/XT | 'Процессор' , г. Воронеж | 25000 | PC/DOS |

*в*

*Рис. 6.3.* Результаты выполнения некоторых операторов реляционной алгебры: *а* — селекция из табл. ПК по условию Наименование = "Нейрон"; *б* — проекция табл. ПК на атрибуты "Наименование" и "Стоимость"; *в* — соединение табл. ПК и ТИПЫ_ОС по условию ПК. Наименование = ТИПЫ_ОС. Наименование (естественное соединение)

таблицах приведены несколько строк, но в реальных базах данных их количество достигает сотен тысяч и миллионов.

Для манипулирования таблицами предложены две системы операторов — реляционная алгебра и реляционное исчисление, базирующиеся на логике предикатов первого порядка. Все реляционные операторы предполагают обращение с целой таблицей как с операндом. Например, оператор селекции реляционной алгебры из одной таблицы, взятой как операнд, производит новую таблицу, составленную из строк первоначальной, удовлетворяющих условию селекции (рис. 6.3). Оператор проекции производит из одной таблицы другую, состоящую из отобранных колонок первоначальной таблицы. Оператор соединения из двух таблиц, взятых как операнды, производит третью, составленную из строк первой таблицы, конкатенированных со строками второй, но только для тех строк, у которых значения атрибутов удовлетворяют условию соединения. Когда условие соединения предполагает равенство значений, а имена атрибутов совпадают, такое соединение называют естественным. Условие соединения в этом случае очевидно, и его указывать не обязательно.

Реляционное исчисление в отличие от алгебры представляет собой не столько систему операторов, сколько свод правил построения корректных формул, описывающих нужные таблицы—результаты. Язык SQL [ДЕЙ88] — это одна из реализаций исчисления, где нет математической символики. В результате получен язык, понятный как программистам, так и пользователям-непрофессионалам. Благодаря многочисленным достоинствам языка

в США его считают даже стандартом для реляционного подхода и используют как средство обращения к базам данных и из прикладных программ, и при работе в диалоговом режиме.

Основной оператор языка называется SELECT (выбрать). Он один реализует одновременно операции селекции, проекции и соединения реляционной алгебры, а также некоторые другие функции. Оператор имеет формат

```
SELECT   < перечень атрибутов >
FROM     < список таблиц >
WHERE    < логическое условие >;
```

где слово FROM означает "из", а WHERE – "где", "при условии, что". Например, рассмотренным несколько выше операторам селекции, проекции и соединения соответствуют следующие три конструкции на языке SQL:

```
SELECT   Наименование, Совместимость, Изготовитель, Стоимость
FROM     ПК
WHERE    Наименование = "НЕЙРОН";

SELECT   Наименование, Стоимость
FROM     ПК;

SELECT   Наименование, Совместимость, Изготовитель, Стоимость,
         Тип_ОС
FROM     ПК, ТИПЫ_ОС;
```

Согласно принятым в языке правилам имена таблиц и атрибутов должны начинаться обязательно с прописной буквы и составлять только одно слово. Когда имя хотят составить из нескольких слов, их соединяют подстрочной чертой. Имена атрибутов можно уточнять именами таблиц, отделяя их друг от друга точками.

Следующий пример иллюстрирует объединение селекции, проекции и соединения в единственном операторе исчисления (рис. 6.4):

```
SELECT   Наименование, Тип_ОС, Стоимость
FROM     ПК, ТИПЫ_ОС
WHERE    ПК. Наименование = ТИПЫ_ОС. Наименование
AND      ПК. Наименование = "НЕЙРОН";
```

где требуется отыскать наименование, типы операционных систем и стоимость персонального компьютера НЕЙРОН.

Сложные логические условия строятся с помощью слов AND ("и"), OR ("или"), либо NOT ("не"), а также круглых скобок, используемых в общематематическом смысле. Условие соединения в этом примере можно было бы опустить, так как речь идет о естественном соединении. Имена таблиц также не обязательно ставить перед каждым именем атрибута. Обычно их используют лишь для того, чтобы избежать двусмысленности или повысить наглядность запросов.

Выполнение запросов, сформулированных с помощью оператора SELECT, обеспечивает реляционная СУБД. Она читает текст запроса, поступивший

к ней из прикладной программы или набранный на экране дисплея, затем выполняет все необходимые действия по манипулированию таблицами базы данных и, наконец, возвращает результат туда, откуда был получен запрос.

| Наименование | Тип_ОС | Стоимость |
|---|---|---|
| НЕЙРОН | НЕЙРОН-ДОС | 15000 |
| НЕЙРОН | MS/DOS | 15000 |
| НЕЙРОН | PC/DOS | 15000 |

*Рис. 6.4.* Результат запроса

**Реляционная СУБД.** Любая СУБД представляет собой комплекс программ, реализующих все функции по управлению базами данных. С ее помощью создают и изменяют базы данных, хранящиеся во внешней памяти ЭВМ, обычно на носителях с прямым доступом (магнитных или оптических дисках). Она осуществляет поиск данных, их копирование и восстановление, управляет одновременным обслуживанием многих пользователей, реализует обмен данными по сетям ЭВМ и множество других функций.

СУБД считается реляционной, если поддерживает таблицы, физические пути доступа к которым скрыты от пользователя; реляционную обработку таблиц хотя бы на уровне операторов селекции, проекции и эквисоединения реляционной алгебры [COD82]. Заметим, что эти ограничения весьма сильны, и далеко не каждая СУБД, которую разработчики считают реляционной, является таковой на самом деле. Однако только на системы, попадающие в указанные рамки, распространяются преимущества реляционного подхода.

Поскольку нас особо интересуют системы, поддерживающие язык SQL, то ограничения Кодда следует усилить еще больше: реляционная СУБД должна поддерживать реляционную обработку таблиц хотя бы на уровне такого оператора SELECT, в котором реализованы по меньшей мере операции селекции, проекции и эквисоединения. Если СУБД удовлетворяет этому ограничению (а SQL — язык стандартный), то ее можно использовать для тесного сопряжения баз знаний с базами данных, реализуемого с помощью описателей объектов. В противном случае подключение базы данных возможно только на уровне обмена отдельными константами, как это делается посредством описателей вопросов.

Для удобства практического использования язык SQL значительно расширен по сравнению с положенным в его основу исчислением предикатов первого порядка. Он наделен множеством функций, не встречающихся в логике предикатов, но необходимых для управления реальными базами данных. Так, в язык введена команда CREATE TABLE (создать таблицу), в результате выполнения которой в базу данных заносится одна первоначально пустая таблица с атрибутами, указанными в команде. Например, для создания таблиц ПК и ТИПЫ_ОС следует задать следующие две команды:

```
CREATE TABLE ПК
                (Наименование  CHAR(10) NOT NULL,
                Совместимость  CHAR(15),
                Изготовитель   CHAR(50),
                Стоимость      INTEGER);
```

```
CREATE TABLE ТИПЫ_ОС
            (Наименование  CHAR(10) NOT NULL,
             Тип_ОС        CHAR(15)).
```

Здесь указаны имена создаваемых таблиц, а также идентификаторы и типы данных каждого их атрибута. Запись CHAR(10) означает, что соответствующий атрибут знаковый с длиной 10 символов, тогда как INTEGER определяет атрибут как целочисленный. Добавка NOT NULL говорит о том, что указанный атрибут не может принимать так называемое нулевое, неопределенное значение. Ее обычно приписывают к тем атрибутам, которые отличают одну строку от другой и называют ключами таблиц.

Для заполнения таблиц данными служит команда INSERT (включить), например:

```
INSERT INTO ПК VALUES          ("НЕЙРОН"; "IBM PC/XT", "ПО
                                им. Королева, г. Киев", 15000);
INSERT INTO ПК VALUES          ("ЭЛЕКТРОНИКА", "IBM PC/XT",
                                "'Процессор', г. Воронеж", 25000);
INSERT INTO ТИПЫ_ОС VALUES     ("НЕЙРОН", "НЕЙРОН–ДОС");
INSERT INTO ТИПЫ_ОС VALUES     ("НЕЙРОН", "MS/DOS");
INSERT INTO ТИПЫ_ОС VALUES     ("НЕЙРОН", "PC/DOS");
INSERT INTO ТИПЫ_ОС VALUES     ("ЭЛЕКТРОНИКА", "PC/DOS");
INSERT INTO ТИПЫ_ОС VALUES     ("ЭЛЕКТРОНИКА", "MS/DOS");
```

где слова INTO и VALUES переводятся как "в" и "значения" соответственно. Набрав на экране подключенного к реляционной СУБД дисплея две команды CREATE и семь приведенных выше команд INSERT, поместим в базу знаний все необходимые данные о персональных компьютерах. После этого обе таблицы можно обрабатывать с помощью других команд языка: SELECT, UPDATE (обновить), DELETE (удалить) и т.п.

Еще одна группа возможностей реляционных СУБД, которая нам понадобится, связана с так называемой концепцией взглядов реляционной модели [ДЕЙ88, ДЕЙ90]. Взгляды еще иногда называют представлениями таблиц. В соответствии с этой концепцией пользователю разрешается объявлять взгляд как результат выполнения команды SELECT и работать с ним так, будто он является хранимой таблицей. Например, в базе данных хранятся две таблицы ПК и ТИПЫ_ОС, а работать с ними будем, как с одной — ТАБЛИЦА_ПК. В данном случае необходимый описатель взгляда имеет вид

```
CREATE VIEW ТАБЛИЦА_ПК
            AS SELECT   Наименование, Совместимость,
                        Изготовитель, Тип_ОС, Стоимость
            FROM        ПК, ТИПЫ_ОС
            WHERE       ПК. Наименование = ТИПЫ_ ОС. Наименование;
```

где команда CREATE VIEW AS означает "создать взгляд ... как...". После выполнения этой команды описание взгляда запоминается в базе данных и к

нему разрешается обращаться так же, как к любой хранимой таблице. Например, в результате выполнения команды

```
SELECT   Наименование, Тип_ОС, стоимость
FROM     ТАБЛИЦА_ПК
WHERE    Наименование = "НЕЙРОН"
```

имеем результат, полученный ранее, когда взгляд не использовался (см. рис. 6.4). Любое упоминание имени взгляда в команде SELECT означает, что СУБД должна сначала построить взгляд, в данном случае — соединить две хранимые таблицы, а затем выполнить эту команду. Применение взглядов существенно сокращает формулировки команд, особенно тех, в которых часто фигурируют одни и те же промежуточные таблицы. Кроме того, взгляды упрощают проектирование базы данных, так как дают возможность сначала составлять необходимые запросы на данные, полагая, что все они хранятся в нескольких огромных таблицах, а затем строить эти таблицы с помощью команд CREATE VIEW. Этот прием представляет собой не что иное, как известное в искусственном интеллекте иерархическое структурирование пространства решений для проблемы проектирования больших баз данных.

## 6.3. РЕЛЯЦИОННОЕ ПРЕДСТАВЛЕНИЕ ОБЪЕКТОВ

Рассмотрим способ более тесного объединения баз знаний и данных, чем допускают описатели вопросов. В результате его применения на базу данных можно переложить хранение некоторых или даже всех описаний объектов предметной области — классов и их элементов. Однако для управления базой данных можно использовать только такую СУБД, которая поддерживает реляционную модель данных и обработку таблиц на уровне оператора SELECT языка SQL.

**Реляционное представление элементов класса.** Когда в базе знаний следует сохранить информацию о многих и многих элементах классов, их можно поместить в реляционную базу данных. Для установления связи элементов с базой знаний в последнюю вводят описатель фиктивного объекта, объединяющий все элементы класса, которые хранятся в базе данных (рис. 6.5). Такой фиктивный элемент обозначают символом @. Будучи единственным, он заменяет все элементы класса, хранящиеся в реляционной базе данных, например элементы класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ, заполненные в таблице с именем ТАБЛИЦА_ПК. Фиктивному объекту в базе данных соответствует одна таблица с тем же именем, строки которой представляют реальные объекты, к примеру объект НЕЙРОН и другие персональные компьютеры (рис. 6.6).

Создавая в базе данных таблицу для фиктивного объекта ТАБЛИЦА_ПК, следует обратить внимание на то, что атрибут "тип_ОС" имеет множество значений. Для каждого многозначного атрибута заводят отдельную хранимую

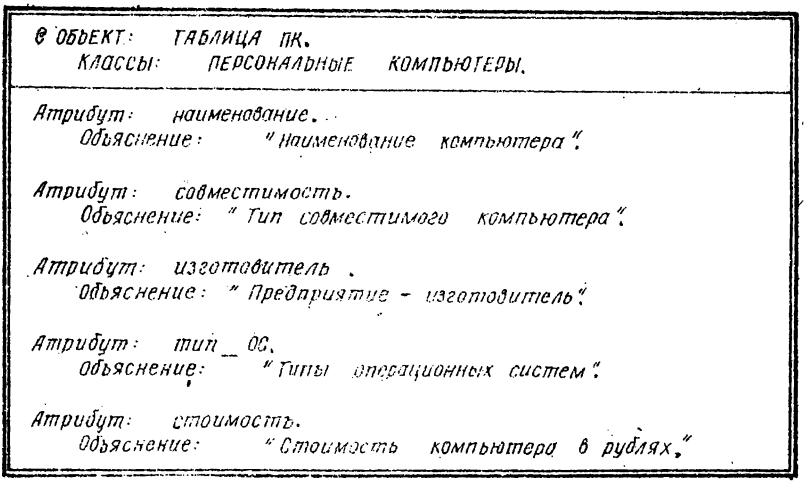



*Рис. 6.5.* Описатель фиктивного элемента класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ


*Рис. 6.6.* Фрагмент базы знаний, в которой один фиктивный элемент заменяет все элементы класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ

таблицу, чтобы обеспечить неизбыточное запоминание данных в так называемой третьей нормальной форме [ДЕЙ90]. Создают все необходимые таблицы с помощью команд CREATE TABLE ПК и CREATE TABLE ТИПЫ_ОС. Для включения сюда строк применяют команды INSERT INTO ПК и INSERT INTO ТИПЫ_ОС. Когда значения какого-то атрибута неизвестны, то в строках таблицы его задают как служебное слово NULL — неопределенную величину [ДЕЙ88], например

```
INSERT INTO ПК VALUES
            ("НЕЙРОН", "IBM PC/XT", "ПО им. Королева, г. Киев",
            NULL);
INSERT INTO ПК VALUES
            ("ЭЛЕКТРОНИКА", "IBM PC/XT", "Процессор, г. Во-
            ронеж", NULL);
```

Далее с помощью команды CREATE VIEW ТАБЛИЦА_ПК строят взгляд, где заказывают соединение двух хранимых таблиц ПК и ТИПЫ_ОС. Если бы существовала хранимая таблица, точно соответствующая описанию фиктивного объекта, то не было бы необходимости заводить взгляд. А когда между хранимой таблицей и описателем класса существуют незначительные расхождения, введение взгляда позволяет выбрать из таблицы необходимые строки и столбцы или просто переставить местами ее колонки.

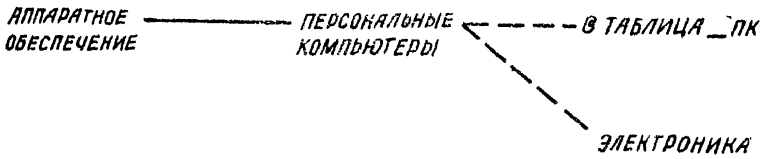



*Рис. 6.7.* Фрагмент базы знаний, где фиктивные элементы сочетаются с реальными

В тех случаях, когда в базе данных сохраняются сведения о каких-либо элементах класса, вовсе не обязательно, чтобы там же запоминалась информация обо всех остальных элементах этого класса. Так, если в описателе элемента ЭЛЕКТРОНИКА класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ встречается такое значение некоторого атрибута, которое невозможно передать с помощью реляционных таблиц (к примеру, заимствование или демон), то этот элемент оставляют в базе знаний. Тогда один элемент класса будет фиктивным, объединяющим все строки реляционной таблицы, а другой — реальным, не имеющим отношения к базе данных (рис. 6.7). За счет применения этого приема в базе знаний можно сохранить все элементы классов из базы знаний, описания которых соответствуют возможностям реляционной модели данных.

**Реляционное представление классов.** Структуры баз знаний системы ЭКСПО строятся так, что основной объем фактической информации сосредоточен в описателях элементов, а не классов. Описатели классов также можно начинить огромным количеством сведений, однако основная их часть заимствуется у элементов классов. Кроме того, именно в описателях классов чаще встречаются конструкции, которые трудно передать с помощью ограниченных возможностей реляционной модели данных. Это прежде всего относится к сложным формам наследования и заимствования значений, использованию демонов и присоединенных процедур в качестве значений атрибутов. Описатели, обладающие такими особенностями, должны целиком или частично сохраняться в базе знаний. Частичное хранение описателя класса в базе знаний означает, что он разбивается на два подкласса, один из которых фиктивный и соответствует части описателя, имеющейся в базе данных, а другой — реальный и объединяет те атрибуты, которые остаются в базе знаний.

Например, описатель класса ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ (рис. 6.8) можно целиком сохранить в базе данных. Его помечают символом @, а в базе данных заводят соответствующие ему следующие хранимую таблицу и взгляд:

```
CREATE TABLE КЛАСС_ПК
                (Мин_стоимость   INTEGER,
                 Макс_стоимость  INTEGER);
CREATE VIEW ПЕРСОНАЛЬНЫЕ_КОМПЬЮТЕРЫ
```

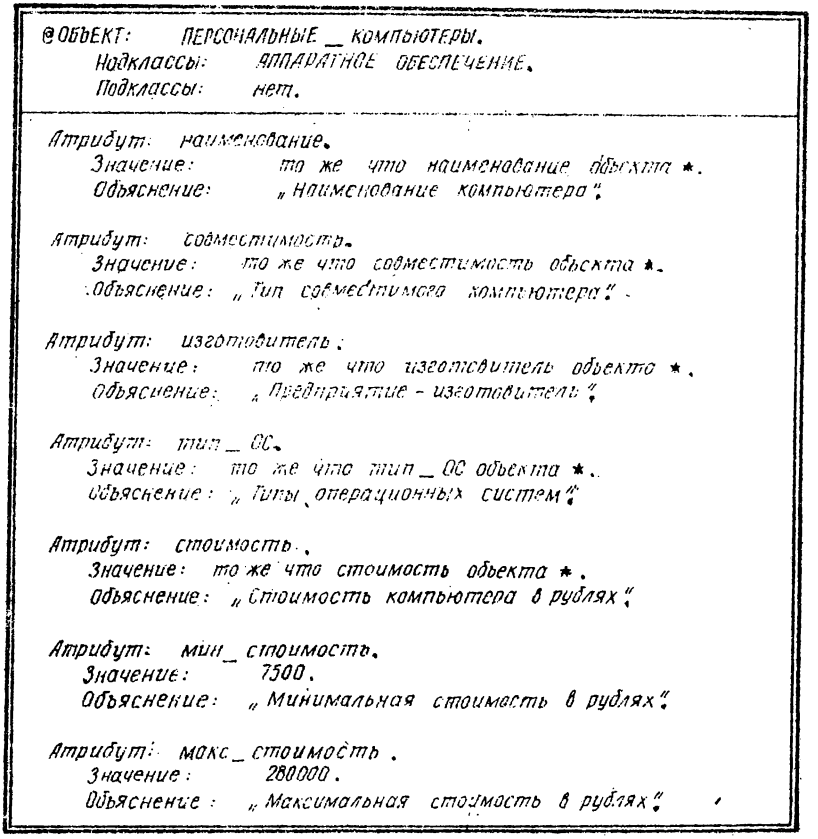



*Рис. 6.8.* Описатель класса в базе данных

```
AS SELECT    Наименование, Совместимость,
             Изготовитель, Тип_ОС, Стоимость,
             Мин_стоимость, Макс_стоимость
FROM         ПК, ТИПЫ_ОС, КЛАСС_ПК;
```

Такое описание взгляда означает, что строки нужной таблицы строятся путем декартова производения таблицы КЛАСС_ПК и результата соединения таблиц ПК и ТИПЫ_ОС, т.е. ранее определенного взгляда с именем ТАБЛИЦА_ПК. В результате декартова произведения двух таблиц каждая строка одной таблицы дополняется поочередно каждой строкой другой таблицы, т.е. выполняется безусловная попарная конкатенация строк обеих таблиц [ДЕЙ88].

Предположим, что все описатели классов, составляющие базу знаний о компьютерах, целиком сохраняются в базе данных, и описатель класса КОМПЬЮТЕРЫ включает информацию из нескольких различных под-

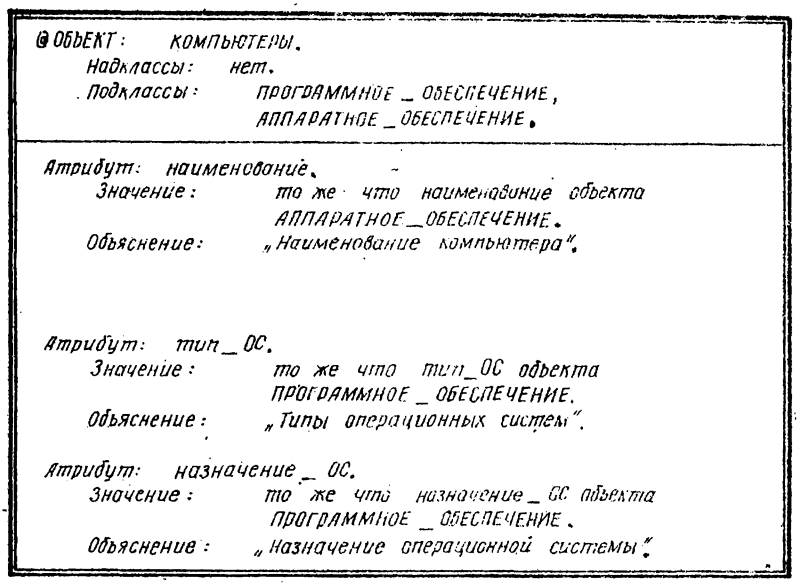



*Рис. 6.9.* Фрагмент описателя класса КОМПЬЮТЕРЫ

классов (рис. 6.9). Такому описателю класса соответствует следующий взгляд:

```
CREATE VIEW КОМПЬЮТЕРЫ
            AS SELECT   Наименование, ...,
                        Тип_ОС, Назначение_ОС
               FROM     АППАРАТНОЕ_ОБЕСПЕЧЕНИЕ,
                        ПРОГРАММНОЕ_ОБЕСПЕЧЕНИЕ
```

Для составления нужного описателя взгляда используют следующие правила:

1) целевой список в фразе SELECT строят из имен атрибутов класса;

2) список FROM составляют из имен всех объектов, встретившихся в описателе класса;

3) в условиях WHERE либо включают совпадающие атрибуты всех классов, попарно соединенные знаком равенства, либо вообще опускают.

Для построения описанного взгляда система управления базой данных автоматически выполнит или соединение указанных таблиц, как для класса КОМПЬЮТЕРЫ, где имеются совпадающие атрибуты, или их декартово произведение, как в предшествующем примере, где совпадающие атрибуты присутствовали только в двух из трех таблиц.

Когда взгляд строят из таблиц, которые сами являются взглядами, то в фразу FROM лучше включать базовые, хранимые таблицы. Это позволит

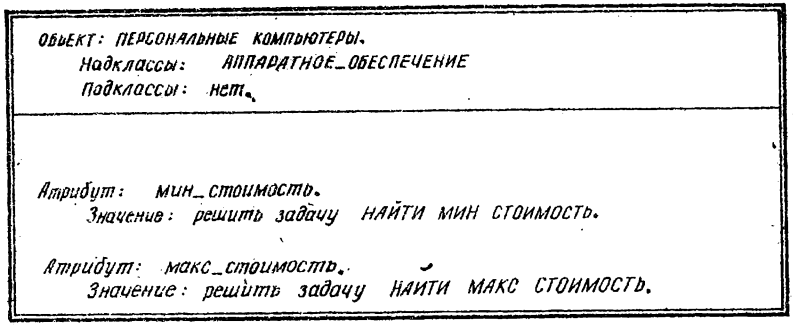



*Рис. 6.10.* Фрагмент описателя класса с использованием присоединенных процедур

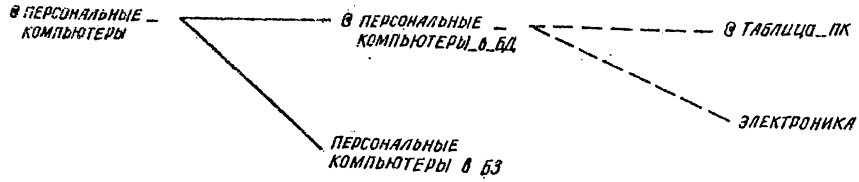



*Рис. 6.11.* Фрагмент базы знаний, где часть описателя класса хранится в базе данных

избежать длинных цепочек переприсваивания одних и тех же данных, так как не все реляционные системы обеспечивают слияние взглядов и их эффективную совместную обработку.

В более сложном случае пусть в описателе класса присутствуют присоединенные процедуры, механизм обработки которых в реляционной модели отсутствует (рис. 6.10). Каждая присоединенная процедура в данном описателе — это задача, которую необходимо решить для отыскания значения минимальной или максимальной стоимости. Такой описатель класса можно разделить на две части: фиктивную, не имеющую элементов, и реальную, к которой приписаны все элементы первоначального класса (рис. 6.11). В результате разделения описателя класса на подклассы часть информации, относящаяся к персональным компьютерам, сохраняется в базе данных (ПЕРСОНАЛЬНЫЕ_КОМПЬЮТЕРЫ_в_БД), а другая часть — в базе знаний (ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ в БЗ).

Как и при работе с элементами классов, на базу данных можно переложить исполнение всех функций, соответствующих ее возможностям, а остальные по-прежнему реализуются непосредственно в базе знаний.

**Организация обмена данными.** Все описания взглядов, участвующие в представлении элементов и классов базы знаний, составляются заранее и заносятся в словарь-справочник, или каталог базы данных. Охватывающая ее система управления базой знаний обеспечивает компиляцию обращений к объектам, ориентируясь на наличие символа '@' в соответствующем описателе. Когда такой символ встретится в описателе, то обращение к объекту

транслируется в последовательность команд на языке SQL; если этот символ отсутствует, то речь идет об обычных операциях работы с базой знаний. Обмен данными между описателями объектов и реляционными таблицами основан на взаимном соответствии атрибутов: первый атрибут описателя — это первый столбец таблицы, второй атрибут — ее второй столбец и т.д.

Технология обработки знаний, реализованная в системе ЭКСПО, предполагает компиляцию описателей, из которых складывается база знаний, в команды, подлежащие выполнению на ЭВМ. При обработке описателей компилятор заменяет тексты на русском языке командами построения физических структур хранения знаний, а все конструкции описателей, в которых предполагается выполнение операций выборки, запоминания или изменения знаний, транслирует в команды обработки соответствующих физических структур. Если строки какого-либо описателя предполагают манипулирование не физической базой знаний, а заменяющими ее реляционными таблицами, то компилятор вместо обычных операций работы с физической базой знаний подставляет тексты команд на языке SQL, специально помечая их. Пометка означает, что эти команды не должны обрабатываться машиной логического вывода системы ЭКСПО, вместо этого их следует передать для исполнения системе управления базой данных.

Проиллюстрируем способ трансляции конструкций базы знаний в команды языка SQL на примере правил логического вывода. Пусть в базе знаний должно быть представлено правило

Если стоимость компьютера превышает 12 тыс. руб.
и стоимость операционной системы для этого компьютера меньше, чем 2 тыс. руб.,
то включить наименование такого компьютера в перечень подходящих компьютеров.

На языке системы ЭКСПО это правило имеет вид

Переменная: подходящий компьютер.
Значение: наименование объекта КОМПЬЮТЕРЫ.
Если: стоимость объекта КОМПЬЮТЕРЫ > 12000
и стоимость_ОС объекта КОМПЬЮТЕРЫ < 2000.

Когда описатель объекта КОМПЬЮТЕРЫ целиком сохраняется в базе данных, т.е. помечен @ , такое правило оттранслируется в следующую команду на языке SQL:

```
SELECT  КОМПЬЮТЕРЫ. Наименование
FROM    КОМПЬЮТЕРЫ
WHERE   КОМПЬЮТЕРЫ. Стоимость > 12000
AND     КОМПЬЮТЕРЫ. Стоимость_ОС < 2000;
```

В условиях сложных правил часто встречаются ссылки более чем на один объект базы знаний, например:

Если стоимость персонального компьютера превышает 12 тыс. руб.

и стоимость операционной системы для этого персонального компьютера меньше, чем 2 тыс. руб.,

то включить наименование персонального компьютера в перечень подходящих компьютеров.

Такому правилу соответствует следующая конструкция:

Переменная: подходящий компьютер.
Значение: наименование объекта КОМПЬЮТЕРЫ.
Если: стоимость объекта ПЕРСОНАЛЬНЫЕ_КОМПЬЮТЕРЫ > > 12000
и стоимость_ОС объекта ОПЕРАЦИОННЫЕ_СИСТЕМЫ < < 2000.

Это правило транслируется в следующую команду:

```
SELECT  ПЕРСОНАЛЬНЫЕ_КОМПЬЮТЕРЫ. Наименование
FROM    ПЕРСОНАЛЬНЫЕ_КОМПЬЮТЕРЫ,
        ОПЕРАЦИОННЫЕ_СИСТЕМЫ
WHERE   ПЕРСОНАЛЬНЫЕ_КОМПЬЮТЕРЫ. Стоимость > 12000
AND     ОПЕРАЦИОННЫЕ_СИСТЕМЫ. Стоимость_ОС < 2000;
```

Построение команды на языке SQL, соответствующей правилу логического вывода, осуществляется компилятором по следующей схеме:

1) целевой список фразы SELECT формируется на основе перечня атрибутов, встретившихся в выводе правила;

2) список таблиц в фразе FROM составляется из имен объектов, фигурирующих в условии правила;

3) условие во фразе WHERE — это видоизмененное условие исходного правила.

Когда в ходе решения задачи обрабатывается любое правило, выполнение действий, которые предписаны его заключением, следующим за словом "то", осуществляется в рамках базы знаний. Машина логического вывода получает из базы данных значения атрибутов, например наименования объектов класса ПЕРСОНАЛЬНЫЕ_КОМПЬЮТЕРЫ, затем выполняет предписанные действия и запоминает результат в рабочей области базы знаний для дальнейшего использования. Существенно, что все результаты выполнения правил, в том числе полученные на основе данных из реляционных таблиц, сохраняются исключительно в базе знаний. Иначе говоря, в команды на языке SQL транслируются только условия правил, их заключения компилируются в обычные операции обмена с рабочей областью базы знаний.

С технической точки зрения не составляет труда транслировать в команды SQL не только условия, но и выводы правил, т.е. сохранять результаты выполнения правил в реляционной базе данных. Однако такой подход на порядок увеличил бы время обработки каждого правила в ходе решения задачи вследствие разницы в скорости обращения к внешней памяти, где находятся реляционные таблицы, и к внутренней, служащей для хранения рабочих областей. За счет раздельной компиляции условий и заключений правил

в системе ЭКСПО преодолена одна из трудностей эффективной обработки знаний, с которой обычно сталкиваются разработчики дедуктивных баз данных [DEL88].

В целом метод реляционного представления объектов основан на разделении базы знаний на две области. Первую составляют компоненты элементов и классов базы знаний, соответствующие возможностям реляционной модели данных. Хранение и обработку сосредоточенных здесь знаний можно целиком переложить на систему управления базой данных. Вторая область включает те знания, особенности которых превышают возможности реляционной модели данных, поэтому их хранение и обработка организуются непосредственно в базе знаний. По соображениям производительности сюда входит рабочая область базы знаний, служащая для сохранения промежуточных и окончательных результатов дедукции. Метод позволяет организовать не только вертикальное разделение базы знаний на области, т.е. выделение целых классов и элементов, подлежащих хранению в базе данных, но и горизонтальное разделение, при котором граница областей проходит через каждый класс или элемент.

Необходимость разделения базы знаний на две взаимодействующие области обусловлена ограниченными возможностями существующих баз данных с точки зрения представления и обработки знаний. По мере совершенствования систем управления базами данных, улучшения их дедуктивных способностей механизм разделения плюс компиляция описателей позволяют естественным путем передвигать границу между областями, перекладывая на базу данных выполнение все новых и новых функций и оставляя за базой знаний решение только задач, превышающих возможности реляционной модели данных.

*ГЛАВА 7*

# РЕАЛИЗАЦИЯ ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ

Определяющее влияние на выбор подхода к реализации интеллектуальной системы оказывают технические характеристики имеющихся программных и аппаратных средств, в среде которых создается и будет позднее работать система. В результате анализа многочисленных факторов, влияющих на трудоемкость реализации и возможности создаваемой системы, для комплекса ЭКСПО в качестве базового аппаратного обеспечения выбрана персональная ЭВМ, совместимая с IBM PC/AT, а программное обеспечение создавалось с помощью средств поддержки языка Пролог.

Среди многочисленных реализаций языка выбор остановлен на Турбо Прологе, и сделано это по следующим важнейшим причинам:

1) наиболее мощная и удобная среда программирования — всевозможные редакторы, отладчики, компоновщики и библиотекари, — причем в сочетании с единой и простой схемой управления этими средствами с помощью спускающихся меню;

2) компилятор наиболее эффективен, генерирует небольшой по объему машинный код и соответственно обеспечивает невысокие затраты времени на исполнение всех программ [АВИ90];

3) велик перечень дополнительных возможностей языка, облегчающих построение всевозможных меню, внешних баз данных, средств обмена текстовой и графической информацией.

Комплекс ЭКСПО состоит из библиотеки, где находятся оттранслированные модули системы, из которых собираются исполняемые программы, а также из двух больших полностью готовых к использованию программ: компилятора баз знаний и основной программы, включающей машину вывода, диалоговый компонент и все остальное, что необходимо для собственно решения задач. В свою очередь, основная программа тоже складывается из двух частей — неизменной, осуществляющей выборку и первичную обработку элементов физической базы знаний, и изменяемой, препрограммируемой надстройки. По желанию пользователя сюда могут быть добавлены дополнительные примитивы дедукции, отсюда исключены какие-то базовые, и таким образом построена новая машина вывода. Входящий в программу диалоговый компонент разрешается дополнять или заменять программами пользователя. Путем компиляции каких-то новых или иной компоновки имеющихся модулей для каждой создаваемой интеллектуальной системы

может быть получена ее собственная версия основной программы. В отличие от нее компилятор всегда остается неизменным, при необходимости он лишь дополняется некоторыми деталями, встречающимися в стандартных описателях знаний, например названиями дополнительных способов логического вывода.

## 7.1. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

На практике процесс построения интеллектуальной системы никогда не выглядит так, что сначала на бумаге создается проект системы, а затем заполняется ее база знаний и компонуется или выбирается готовая основная программа. Обычно создание готовой к эксплуатации системы растягивается на несколько итераций, включающих построение прототипа, скелета будущей системы и целую серию уточнений, дополнений прототипа, пока он не удовлетворит предъявляемым требованиям. В результате изменение и компилирование базы знаний, а также перекомпоновку основной программы повторяют несколько раз, постепенно приближаясь к нужному результату.

**Компилятор базы знаний.** Пользователь строит, изменяет и компилирует описатели базы знаний с помощью диалоговой программы, управляя ее работой посредством спускающихся меню (рис. 7.1). Сначала он должен "выбрать" нужный элемент, отмечая в нескольких вертикальных меню, спускающихся из этого гнезда, файл базы знаний, тип элемента (правило, объект, вопрос и т.п.) и его идентификатор (например, П—032). Затем следует прочитать элемент из базы знаний либо, когда он новый, ввести с клавиатуры в окно редактора, занимающее основную часть экрана дисплея. Для ввода и изменения текста описателя служат команды редактирования, реализующие полный набор операций работы с текстами, включая работу с блоками текста, несколькими окнами и файлами. Самое первое меню, опускающееся из гнезда "изменить", сверху вниз задает последовательность действий по изменению выбранного элемента: чтение на экран из базы знаний или копирование из текстового файла, редактирование, запись элемента в базу знаний, а при необходимости также печать и удаление элемента.

Компиляция элемента выполняется во время помещения описателя в базу знаний. На этом шаге делается полный грамматический разбор описателя и отыскиваются всевозможные синтаксические ошибки. Только после того, как выявлены и исправлены все неточности, элемент помещается в базу знаний, а на экран в специальное окно сообщений выводится подтверждение записи.

Когда хотя бы один элемент базы знаний изменялся (был удален, добавлен или откорректирован), файл, в котором произошло изменение, помечается как несогласованный. Считается, что в таком файле, хотя все его элементы и прошли синтаксический контроль, еще возможны семантические ошибки. Это наиболее серьезные ошибки проектирования базы знаний, которые обычно обнаруживаются лишь после длительного тестирования или

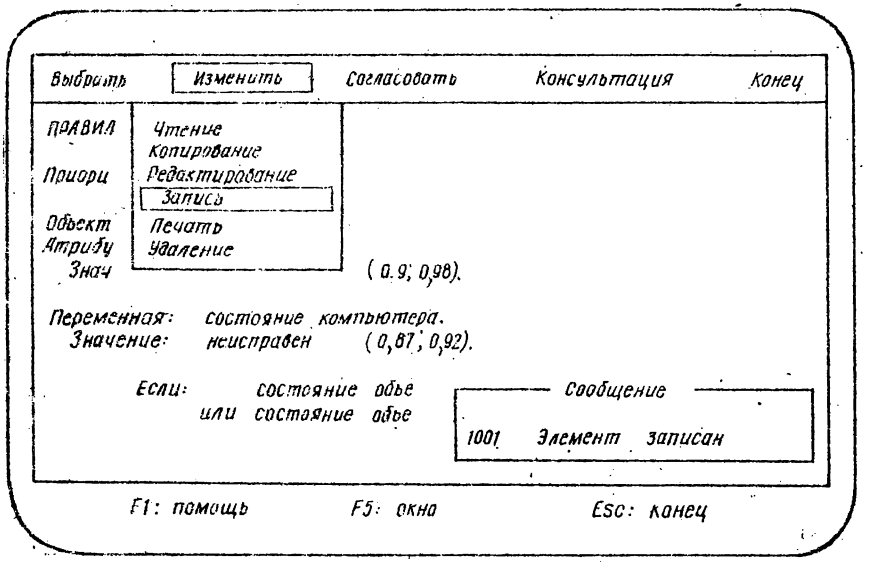



*Рис. 7.1.* Экран дисплея при компиляции базы знаний

даже на этапе опытной эксплуатации интеллектуальной системы. Компилятор резко сокращает, а во многих случаях даже сводит к нулю затраты на этот вид деятельности. Для этого в него встроены специальные алгоритмы, обследующие базу знаний и умеющие обнаруживать подобные ошибки, прежде всего:

наличие тупиковых ветвей рассуждений, т.е. цепочек вывода, ни при каких сочетаниях входных параметров не приближающих решения проблемы;

наличие разрывов в цепочках рассуждений, когда, например, какое-либо значение переменной необходимо для решения задачи, но оно нигде логически не выводится и не запрашивается у пользователя.

Практика показала, что обе ошибки чаще всего встречаются одновременно по причине того, что проектировщик базы знаний ошибся при задании имен некоторых переменных, атрибутов или констант. Все эти ситуации, как и многие другие, без труда обнаруживаются компилятором и могут быть немедленно исправлены на экране с помощью встроенного редактора.

Для обнаружения и исправления такого рода ошибок базу знаний необходимо "согласовать", выбирая соответствующее гнездо горизонтального меню после каждой серии изменений. Сообщения о результатах семантического анализа сразу выдаются на экран дисплея, но в отличие от синтаксических ошибок эти сообщения носят рекомендательный характер и не обязательны для исполнения. Это сделано из-за того, что база знаний может находиться в процессе построения и в ней не всегда присутствуют все необходимые элементы. Если в ответ на сообщение компилятора проектировщик подтвердит, что обнаруженные семантические несоответствия им предусмот-

рены, то файл базы знаний помечается как согласованный и может быть передан машине вывода.

Следующее по порядку гнездо горизонтального меню запускает тестирование базы знаний путем загрузки и выполнения основной программы. Здесь проверяют правильность размещения текстов вопросов, результатов и другой информации в окнах диалога и меню, последовательность вопросов, корректность результатов и т.п. Если полученные выводы отличаются от ожидаемых, то, задавая вопросы, "как" выведен тот или иной результат, можно обойти все цепочки рассуждений и отыскать логическую ошибку. Для ее исправления следует в любом месте диалога ввести слово "конец", чтобы немедленно вернуться в окно редактора, вызвав туда необходимый элемент базы знаний. Возврат из основной программы возобновляет работу компилятора. После изменения всех необходимых описателей цикл согласования и тестирования повторяют. Последнее гнездо горизонтального меню завершает работу программы и убирает с экрана все окна.

Применение языка Пролог как инструмента построения компилятора существенно упростило разработку и отладку программ. Наиболее важная и обычно трудоемкая их часть, связанная с трансляцией описателей в конструкции физической базы знаний и наоборот на самом деле оказалась чуть ли не наименьшей по объему. Соответствующая ей программа, по сути, свелась к записи на языке логики правил формальной грамматики языка описателей, доказательство же того, принадлежат ли поступающие строки символов этому языку, полностью взял на себя интерпретатор Пролога. Это позволило высвободить ресурсы для улучшения другой очень важной части компилятора, где проводится семантический анализ базы знаний, что также не составило большого труда, поскольку и здесь понадобилось лишь описать логические требования к непротиворечивой и согласованной базе знаний, а их проверку взял на себя стандартный интерпретатор. В результате получен компилятор, где полный синтаксический контроль описателей сочетается с семантическим анализом базы знаний, что существенно облегчает построение интеллектуальных систем, позволяя за счет расширения рамок контроля сократить затраты на эту весьма трудоемкую деятельность. Кроме того, хронологически компилятор создавался самой последней программой комплекса ЭКСПО, после того как было выяснено содержимое всех описателей базы знаний. Часть чисто исследовательских работ здесь невелика, и особое внимание уделялось оценке затрат всех ресурсов на построение и отладку входящих сюда программ. Оказалось, что полная трудоемкость работ над компилятором, начиная от проектирования программ и заканчивая их тестированием на реальных примерах, не превысила 25 человеко-дней! И хотя в это число не входят затраты на последующие мелкие корректировки программ, в основном обусловленные уточнением синтаксиса описателей, полученный результат еще раз подтверждает высокие потенциальные возможности логического программирования и Пролога в области искусственного интеллекта и при решении таких традиционных задач, как построение компиляторов.

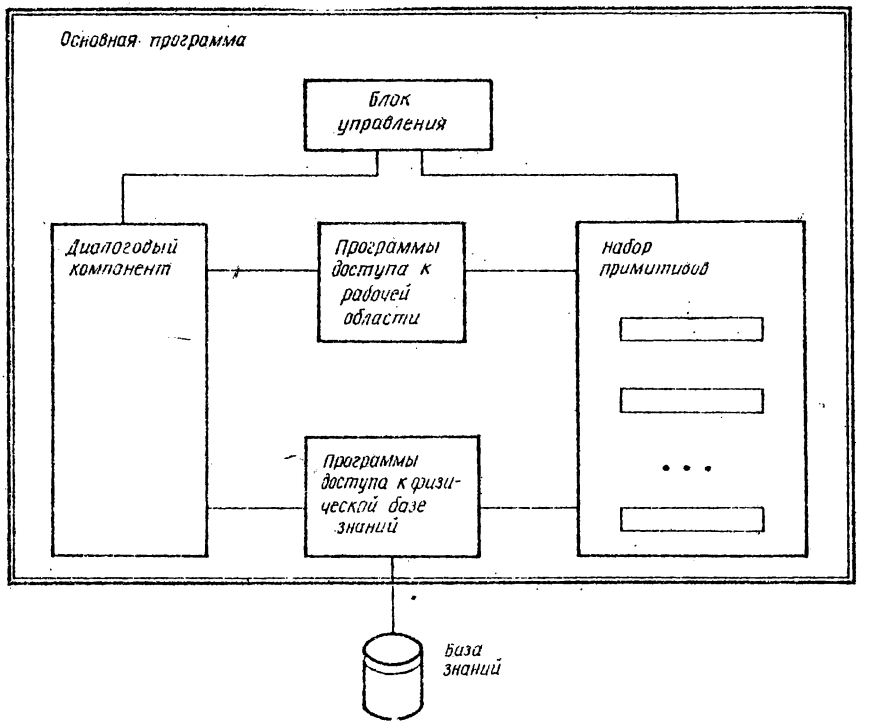



*Рис. 7.2.* Архитектура основной программы

**Основная программа.** Если попытаться обособить функции, которые исполняет основная программа, в которой объединены главным образом машина вывода и диалоговый компонент, то они попадут в одну из трех больших групп: 1) методы логического вывода, реализованные как примитивы дедукции, 2) организация диалога с помощью меню и обмена текстовой, а также графической информацией, 3) средства обработки физической базы знаний и рабочих областей (рис. 7.2).

Кроме того, имеется небольшой блок управления, под контролем которого решаются задачи. Работу именно этого блока пользователь программирует, составляя описатели проблем и задач. Точнее, эти описатели компилируются в более мелкие команды, а уже их исполняет блок управления. Он начинает выполнение того или иного примитива дедукции, инициирует выдачу пользователю вопросов и чтение ответов через диалоговый компонент, пускает вывод объяснений и исполняет другие связанные с управлением функции.

Работа диалогового компонента также программируется непосредственно пользователем, когда он составляет описатели вопросов. Эти описатели позднее компилируются в мелкие команды, с которыми имеют дело программы, составляющие этот компонент.

Еще один небольшой блок обслуживает обмен с рабочей областью базы знаний. Он выполняет запросы на чтение, запись и удаление фактов из рабочей области, получая команды от других блоков. Такую же работу делает блок доступа к физической базе знаний, выполняя те же самые команды, но применительно к файлам, расположенным во внешней памяти ЭВМ. Команды к нему тоже поступают от других блоков.

Блоки, составляющие основную программу, представляют собой подпрограммы разного размера. Нетрудно заметить, что все они работают по одной и той же схеме, получая извне последовательность команд и реализуя их исполнение. Блок управления и диалоговый компонент получают команды из описателей, т.е. в конечном счете от пользователя, роль команд для примитивов дедукции играют конструкции базы знаний, также составленные пользователем как описания объектов и правил, а блоки доступа к внешней базе знаний и к рабочей области получают команды от других программ. Соответственно и алгоритмы работы всех блоков выглядят одинаково — как последовательность правил вида: если < команда > , то < действие > , а это именно конструкция, являющаяся в Прологе основной. Набор таких конструкций, записанных на языке логики, и составляет программу каждого блока. Эта особенность языка упростила построение стандартных компонентов системы и облегчила подключение к ним подпрограмм пользователей, расширяющих или заменяющих некоторые элементы основной программы. Дополнительные команды и соответствующие им подпрограммы пользователей подключаются через блок управления. Для этого ему сообщаются название команды и имя внешней подпрограммы, реализующей ее исполнение. Блок управления, встречая добавленную команду, всегда передает управление подключенной программе, а не стандартному компоненту.

Отдельные функции, предписанные командами действий, могут быть реализованы на любом языке программирования, подчиняющемся стандартным соглашениям о связях внешних программ. И в этой части системы Пролог является удобным инструментом реализации вследствие присущей ему легкости построения и манипулирования внешними базами данных, естественности обработки текстовой и графической информации. Однако наиболее важным является то, что на нем удобно описывать разнообразные методы логического вывода, пополняя тем самым набор стандартных примитивов дедукции в библиотеке системы. Многие считают, что Пролог — не столько язык обычного программирования, сколько развитое инструментальное средство создания интеллектуальных, в том числе экспертных систем. В данном случае это означает, что, работая с Прологом, пользователь настраивает инструментальную систему не путем трудоемкого создания традиционных программ, а используя другое высокоуровневое инструментальное средство.

Однако существует распространенное заблуждение, что раз интерпретатор Пролога сам использует простой метод вывода, известный в логике как нисходящий поиск в глубину слева направо, то с его помощью трудно реализовать иные способы дедукции, например поиск в ширину или прямую

цепочку рассуждений. Здесь смешивают способ реализации языка и функции написанных на нем программ. Встроенный интерпретатор Пролога — это лишь средство, призванное облегчить работу программиста и повысить его производительность. Пользуясь этим средством, можно строить программы, работающие не так, как стандартный интерпретатор. В учебниках по программированию на Прологе можно отыскать несколько подходов к построению различных методов дедукции, таких, как обратная цепочка в глубину [БАК92, БРА90, КЛА87], обратная цепочка в ширину [БРА90], прямая цепочка [БАК92]. В этих и других источниках не раз рассматривались способы управления диалогом, организации внешних баз данных и обмена с рабочей областью. Кроме того, в работе [БАК92] дан анализ нескольких подходов к реализации логического вывода на Прологе, а также алгоритмы и основные фрагменты программ, составляющих примитивы дедукции системы ЭКСПО.

**Обмен данными между программами.** Компилятор и основная программа в процессе построения и работы интеллектуальной системы обмениваются информацией через общую базу знаний. Блоки основной программы передают друг другу данные через рабочую область, а также в виде параметров соответствующих подпрограмм. И в том, и в другом случаях данные организованы в сложные конструкции языка Пролог. А поскольку в состав основной программы разрешается включать модули, написанные на других алгоритмических языках, программисту необходимо знать, что стоит за конструкциями Пролога и какие структуры данных соответствуют им в обычных языках. Проиллюстрируем принципы обмена данными на примере рабочей области, которая строится в оперативной памяти ЭВМ как динамически изменяемый набор безусловных утверждений Пролога. Каждое имеющееся здесь утверждение констатирует один элементарный факт, определяет одно значение атрибута или некоторой переменной. В программе на Прологе все факты сохраняются в так называемой базе данных, допускающей динамическое наращивание и всевозможные изменения утверждений. Утверждения, составляющие рабочую область, имеют один и тот же вид fact (VAL, *Pmin*, *Pmax*, Tlist), где параметр VAL определяет значение переменной или атрибута, числа *Pmin* и *Pmax* задают границы интервала надежности факта, а Tlist представляет собой список внутренних идентификаторов термов базы знаний, использованных для логического вывода данного факта (он нужен для объяснений).

Фрагмент программы, где объявляются эти параметры, на Турбо Прологе выглядят следующим образом:

```
DOMAINS
    VAL          = oa(O, A, V); a(A, V)
    O, A         = string
    V            = i(integer); r(real); s(string)
    Pmin, Pmax   = real
    Tlist        = ID*
    ID           = integer
```

Параметр VAL представляет собой функцию либо трех, либо двух аргументов. Функция с именем *oa* имеет три аргумента, где запоминаются имена объекта *O* и атрибута *A*, а также одно значение *V* данного атрибута. Два аргумента функции *a* служат для хранения соответственно имени переменной *A* и значения *V*. Следующее определение констатирует, что имена объектов *O* и атрибутов или переменных *A* представляют собой строки символов стандартного типа string. Значение атрибута и переменной может быть либо целым, и тогда его задают аргументом функции *i*, либо действительным числом, которому соответствует функция с именем *r*, либо строкой знаков (функция *s*). Числа *P*min и *P*max — действительные величины, а Tlist считаем списком целочисленных идентификаторов термов. Например, два факта "стоимость объекта НЕЙРОН — — 15000" и "состояние компьютера — — неисправен [0,7; 0,9]", выведенные с помощью ряда правил, запоминаются в рабочей области в таком виде:

```
fact(oa("НЕЙРОН", "стоимость", i(15000)), 1, 1, [36, 27, 19, 3]).
fact(a("состояние компьютера", s("неисправен")), 0.7, 0.9, [18, 24]).
```

Списки чисел в квадратных скобках — внутренние идентификаторы термов, присвоенные компилятором во время трансляции описателей правил. Они короче внешних имен и используются программами доступа к базе знаний для быстрой выборки термов. В данном случае первый факт выведен четырьмя правилами, а второй — подтвержден двумя.

Для выполнения операций с рабочей областью служат три глобальных предиката Пролога:

```
add_fact(VAL, Pmin, Pmax, ID)   —  добавление одного факта;
find_fact(VAL, Pmin, Pmax, Tlist) —  чтение факта;
del_fact(VAL)                   —  удаление факта.
```

В традиционных языках программирования, таких, как Си или Паскаль, глобальным предикатам соответствуют внешние продуценты или функции. Для обращения к нужному предикату достаточно включить в программу на одном из этих языков вызов процедуры add_fact, find_fact или del_fact. Однако для того чтобы инструментальная система выполнила заказанную работу, следует правильно организовать передачу параметров от одного языка к другому.

Целому числу (тип integer Пролога) в других языках соответствует одно слово длиной 2 байта, действительному (real) — 8 байтов, а строке знаков (string) — набор пар < символ, указатель на следующий >. Эти типы данных считаются простыми, они встречаются во всех языках, и их передача выполняется напрямую, без каких-либо преобразований.

Список Пролога представляется в памяти как набор элементов < функтор списка, элемент списка, указатель на следующий >, где первый и последний элементы имеют длины 1 и 4 байта соответственно. Например, на языке Си список Tlist может быть объявлен так:

```
Struct ilist {
    unsigned char functor;      /* функтор списка */
    int id;                     /* элемент списка */
    struct ilist * next;        /* указатель      */
    }Tlist;
```

Функциям Пролога в языке Си ставят в соответствие структуры. Первый элемент каждой структуры содержит имя функции, т.е. функтор, далее по порядку следуют все аргументы. Например, для параметра V первому функтору i в памяти соответствует 1; второму r — 2; третьему s — 3. Нужное объявление записи для параметра V на языке Си выглядит как

```
typedef struct ilist {
    unsigned char functor;
    union{
    int i;
    real r;
    char * s;
    }
  }V;
```

Соответствие между более сложными конструкциями Пролога и структурами данных других языков программирования устанавливается по тем же самым принципам, поскольку все сложные конструкции строятся из одних и тех же простых элементов — переменных, констант, списков и функций. Одни и те же способы согласования структур применимы для конструкций рабочей области и физической базы знаний, где хранятся скомпилированные описатели, которые также понадобятся разработчику интеллектуальной системы, если он принял решение написать собственную программу управления диалогом или новый примитив дедукции.

## 7.2. ОРГАНИЗАЦИЯ ФИЗИЧЕСКОЙ БАЗЫ ЗНАНИЙ

База знаний, хранящаяся во внешней памяти ЭВМ на носителях с прямым доступом, имеет логический и физический уровни представления (рис. 7.3). База знаний на логическом уровне соответствует тому, как ее видят пользователь и инженер по знаниям. На этом уровне существуют описатели объектов, правил, задач и т.д. Структуры данных, используемые для хранения элементов знаний во внешней памяти ЭВМ, в совокупности образуют физический уровень представления. Если логическое представление ориентировано на потребности человека, имеющего дело с интеллектуальной системой, то физическое — на эффективную обработку знаний компьютером. Компилятор базы знаний, транслируя конструкции логического уровня в физические и обратно, обеспечивает согласование обеих мотивировок. Существенно, что благодаря использованию компилятора ни пользователь, ни инженер по знаниям никогда не взаимодействуют непосредственно с физи-

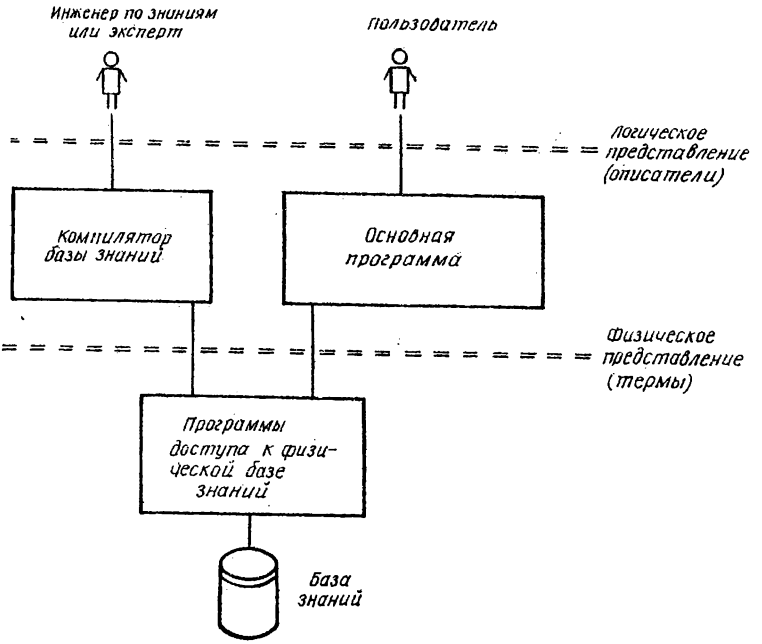



*Рис. 7.3.* Два уровня представления знаний

ческими структурами (списками, ссылками, битами, байтами и т.п.). Их интерпретируют только программы инструментальной системы, а общение с пользователями всегда ведется в терминах описателей, составленных на естественном языке.

**Термы.** Когда компилятор обрабатывает описатель вопроса или объекта, он порождает соответствующую этому описателю последовательность термов физической базы знаний. Независимо от того, какой описатель — правила или объекта — использовался для порождения терма, все термы имеют одинаковую структуру

term(ID, Alist, Clist, Tid, Uid),

где ID — внутренний (цифровой) идентификатор терма; Alist — список предполагаемых термом действий, команд; Clist — логическое условие; Tid — внутренний идентификатор текста, приписанного терму в описателе; Uid — внешний идентификатор терма.

Все внутренние идентификаторы имеют тип integer. Они строятся компилятором и используются для быстрого поиска в физической базе знаний. Внешние идентификаторы задает пользователь, присваивая имена атрибутам, описателям правил и объектов. Например, внутреннему идентификатору 200 может соответствовать имя правила П—032, тогда как идентификатору 210 — имя атрибута "стоимость" объекта НЕЙРОН. Параметр Tid содер-

жит ссылку на текст объяснения, заданного пользователем в той части описателя, откуда получен терм.

Список Alist описывает последовательность действий, которые должна совершить машина вывода, если условие терма окажется истинным. Список строится из следующих команд:

st(A, V, X, Y, IDlist) — присвоить переменной или атрибуту A значение V, учитывая, что сила правила задана как X и Y;

del(A, IDlist) — удалить из рабочей области все значения переменной или атрибута A;

del1(A, V, IDlist) — удалить значение V переменной или атрибута A;

same(A1, A2, X, Y, IDlist) — атрибуту или переменной A1 присвоить те же значения, которые имеет атрибут или переменная A2, учитывая силу правила X и Y;

add(A, V, IDlist)
sub(A, V, IDlist)
mul(A, V, IDlist)
div(A, V, IDlist)

соответственно увеличить, уменьшить, умножить и разделить значение переменной или атрибута A на константу или переменную V;

task(Name) — решить задачу с именем Name.

Список Clist есть необходимое условие истинности терма. Он строится из команд сравнения:

eq(A, V, IDlist) — переменная или атрибут A равны V;

ne(A, V, IDlist) — переменная или атрибут A не равны V;

known(A, IDlist) — значения переменной или атрибута A известны;

notknown(A, IDlist) — значения переменной или атрибута A неизвестны;

l (A, V, IDlist)
le(A, V, IDlist)
g (A, V, IDlist)
ge(A, V, IDlist)

значение переменной или атрибута A соответственно меньше, меньше или равно, больше, больше или равно значению V.

Список IDlist, встречающийся в условиях и в предписываемых термами действиях, строится компилятором базы знаний. Когда он стоит в элементах списка Alist, то содержит внутренние идентификаторы термов, где используется результат выполнения данной команды; в элементах списка Clist он содержит идентификаторы всех термов, откуда может быть получено данное значение переменной или атрибута.

В ходе компиляции любого описателя из него прежде всего извлекаются все длинные текстовые строки. Тексты сохраняются в базе знаний отдельно от термов, доступ к ним осуществляется по внутренним идентификаторам. Такая организация физической базы знаний существенно ускоряет обработку термов, а именно в них сосредоточена вся необходимая машине вывода информация. Выборка длинных текстовых строк производится по внутренним идентификаторам только тогда, когда они действительно необходимы основной программе. К примеру, тексты правил на естественном языке читаются лишь при объяснениях.

**Компиляция описателей правил и объектов.** Обрабатывая любой описа-

тель правила, компилятор строит для него один терм и при наличии текста правила на естественном языке заводит один текстовый элемент.

Пусть имеется правило с внешним идентификатором П—032:

```
Объект          состояние.
Атрибут:        ПРОЦЕССОР.
    Значение:   неисправен (0,9; 0,98).
Переменная:     состояние компьютера.
    Значение:   неисправен (0,87, 0,92).
    Если:       состояние объекта ПЛАТА ПРОЦЕССОРА — — неисправна
                состояние объекта ПЛАТА ПАМЯТИ     — — несправна.
Объяснение:
    "Если       плата процессора неисправна
      и         плата памяти неисправна
     то         процессор неисправен (0,9; 0,98)
      и         компьютер неисправен (0,87; 0,92).
    Дата:       15.04.91.
    Автор:      Иванов А.Б.".
```

Сначала для этого правила компилятор построит один элемент в хранилище текстов базы знаний. Пусть внутренний идентификатор текста будет 100:

text(100, "Если плата процессора. . . Автор: Иванов А.Б.").

Затем в физическую базу знаний помещаем следующий терм с идентификатором, например, 200:

```
term (200,  [st (oa ("ПРОЦЕССОР", "состояние"), s ("неисправен")), 0.9,
             0.98,  [119, 12]),
             st (a ("состояние  компьютера"),  s  ("неисправен")),  0.87,
             0.92, [120, 138, 2])],
             [eq (oa ("ПЛАТА ПРОЦЕССОРА", "состояние"), s ("неисправна"), [86, 88]),
             [eq (oa ("ПЛАТА  ПАМЯТИ",  "состояние"),  s ("неисправна"), [34])],
             100, r ("П—032")).
```

В квадратных скобках содержатся списки внутренних идентификаторов других термов, где встречаются те же значения переменной и трех атрибутов. Внешний идентификатор правила задан как аргумент функции r, указывающей на то, что здесь терм получен из правила, а не из описателя какого-то объекта.

Каждый описатель объекта в отличие от правила транслируется в несколько термов. Количество термов, построенных компилятором, совпадает с числом атрибутов в описателе, причем каждому способу присваивания значений атрибуту соответствует свое содержимое терма.

В самом простом случае значение атрибута задается константой, например:

ОБЪЕКТ: НЕЙРОН.

. . .

Атрибут: стоимость.
Значение: 15000.
Объяснение: "Стоимость компьютера в рублях".

Это описание атрибута оттранслируется в следующие терм и строку текста:

```
term (210,  [st (oa ("НЕЙРОН", "стоимость"),  i (15000), 1, 1, [76] )],
            [ ],
            110, oa ("НЕЙРОН", "стоимость") ).
text (110,  "Стоимость компьютера в рублях").
```

Список Clist в данном случае не нужен, поэтому он пуст.

Если значение атрибута заимствуется у переменной или другого объекта, как в показанном ниже описателе:

ОБЪЕКТ: ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ.

. . .

Атрибут: стоимость ОС.
Значение: то же что стоимость объекта ОПЕРАЦИОННЫЕ СИСТЕМЫ.,

то в терме появляется команда same:

```
term (211,  [same (oa ("ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ", "стоимость
                                                    ОС"),
                   oa ("ОПЕРАЦИОННЫЕ СИСТЕМЫ", "стоимость"),
                   1, 1, [34, 67] )],
            [ ],
            –1, oa ("ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ", "стоимость ОС") ).
```

Поскольку здесь нет объяснения, то текстовый элемент не заводится, а на позиции его идентификатора в терме появляется отрицательное значение, говорящее о том, что выборка текста из базы знаний не нужна.

Еще один способ задания значений атрибутам заключается в использовании присоединенной процедуры:

ОБЪЕКТ: ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ.

. . .

Атрибут: макс стоимость.
Значение: решить задачу НАЙТИ МАКС СТОИМОСТЬ.

Такой конструкции в базе знаний соответствует терм

```
term (212,  [task ("НАЙТИ МАКС СТОИМОСТЬ") ],
            [ ],
            –1, oa ("ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ", "макс стоимость") ).
```

После того как компилятор обработает все описатели, он передает последовательность термов модулям доступа к физической базе знаний для

записи на внешний носитель. Модули помещают термы в структуры хранения знаний, попутно создавая другие структуры — пути доступа к данным, служащие для быстрого поиска термов. Позднее те же самые модули обеспечивают чтение термов по командам, поступающим от компилятора и из основной программы. При этом заблаговременно построенные пути доступа используются для быстрой выборки термов из структур хранения.

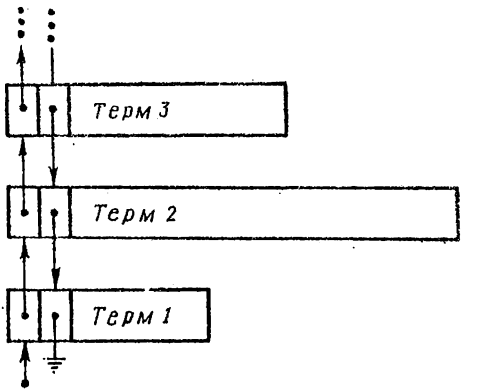



*Рис. 7.4.* Двусвязный список с элементами переменной длины

**Структуры хранения и пути доступа.** Понятие физической базы знаний объединяет структуры хранения знаний и пути доступа к ним. В качестве структур хранения, предназначенных непосредственно для запоминания термов во внешней памяти ЭВМ, служат организованные на внешнем носителе списки, а пути доступа к элементам списков имеют вид B-деревьев [ТИО85].

Списки строятся из термов, в которые компилятор транслирует описатели базы знаний (рис. 7.4). Каждому терму соответствует один элемент списка. Длина как терма, так и списка в целом не ограничивается.

Использование списковых, а не каких-либо иных структур вызвано тем, что базе знаний, вообще говоря, свойственна высокая частота изменений, обусловленных непрерывным пополнением, корректировками и удалениями знаний. Кроме того, любой элемент знаний имеет заранее непредсказуемую длину. Согласование частых изменений и неопределенной длины записей лучше всего достигается в списковых структурах с витруальной адресацией элементов, где виртуальный адрес определяет не физическое расположение элемента, а скорее номер терма, позиции, где хранится реальный адрес. Такая двухступенчатая адресация позволяет внутри списка локализовать изменения любых его элементов, не требуя корректировки виртуальных адресов (рис. 7.5).

Для быстрого поиска любого элемента списка служат B-деревья. Это разновидность сбалансированных деревьев с гарантированно низкой верхней границей времени отклика [ТИО85]. Здесь сохраняются идентификаторы термов вместе с виртуальными адресами соответствующих элементов списка. Дерево обеспечивает гарантированно быстрый перевод идентификаторов в виртуальные адреса, а механизм обработки списка дальше переводит эти адреса в реальные, указывающие местоположение записей на устройстве памяти. Многоступенчатая схема адресации необходима для того, чтобы поддерживать на неизменно высоком уровне скорость выборки постоянно изменяющихся элементов базы знаний.

Программы создания и обработки списков и B-деревьев в совокупности

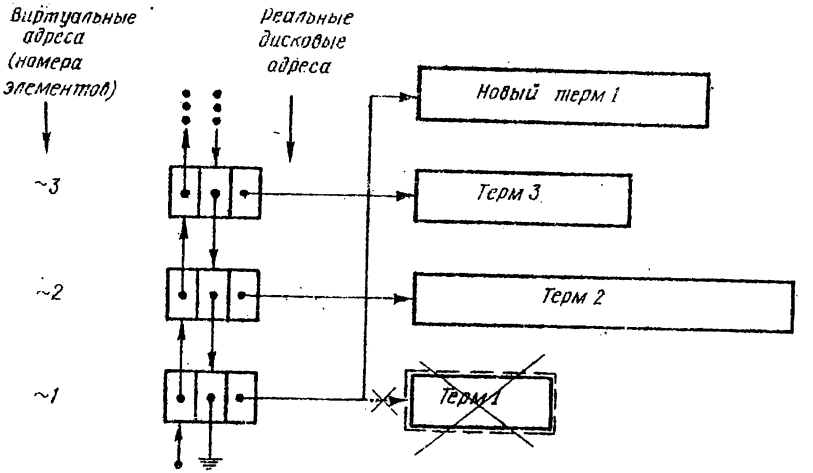



*Рис. 7.5.* Замена одного элемента списка новым

реализуют все необходимые операции обработки термов по их внутренним идентификаторам. Типичные команды, которые выполняют программы доступа к физической базе знаний, — построить B-дерево, создать списковый файл, удалить, изменить или добавить один элемент списка, а также быстро отыскать элемент по его внутреннему идентификатору.

**Модули базы знаний.** База знаний по желанию проектировщика может состоять из одного либо нескольких файлов. Каждый файл имеет одну и ту же структуру и включает несколько списков и B-деревьев (рис. 7.6). Все структуры, входящие в состав файла, строятся автоматически, без вмешательства пользователя. От него требуется лишь, пользуясь меню компилятора, указать имя файла, куда должны помещаться элементы базы знаний. Обычно в одном файле объединяют все элементы, имеющие отношение к некоторой достаточно большой задаче, и такой файл называется модулем базы знаний. Вся база знаний может складываться из нескольких таких модулей.

Разделение базы знаний на модули выгодно по многим причинам. Во-первых, повышается эффективность обработки больших баз знаний. В любой интеллектуальной системе, когда она решает задачи, многократно повторяются одни и те же операции сначала поиска нужных в данный момент правил и объектов, а затем их обработки. Чем меньше размер пространства, где производится поиск, тем быстрее будет найдено каждое правило. Однако чем больше правил содержится в базе знаний, тем, вообще говоря, обширнее знания системы и выше уровень ее компетентности. Разделение проблемы на задачи и использование модулей позволяют согласовать эти противоречивые требования. Если все элементы базы знаний, имеющие отношение к текущей задаче, сосредоточить в одном модуле, то только здесь следует искать необходимые термы и никакие другие модули при этом задействовать не следует.

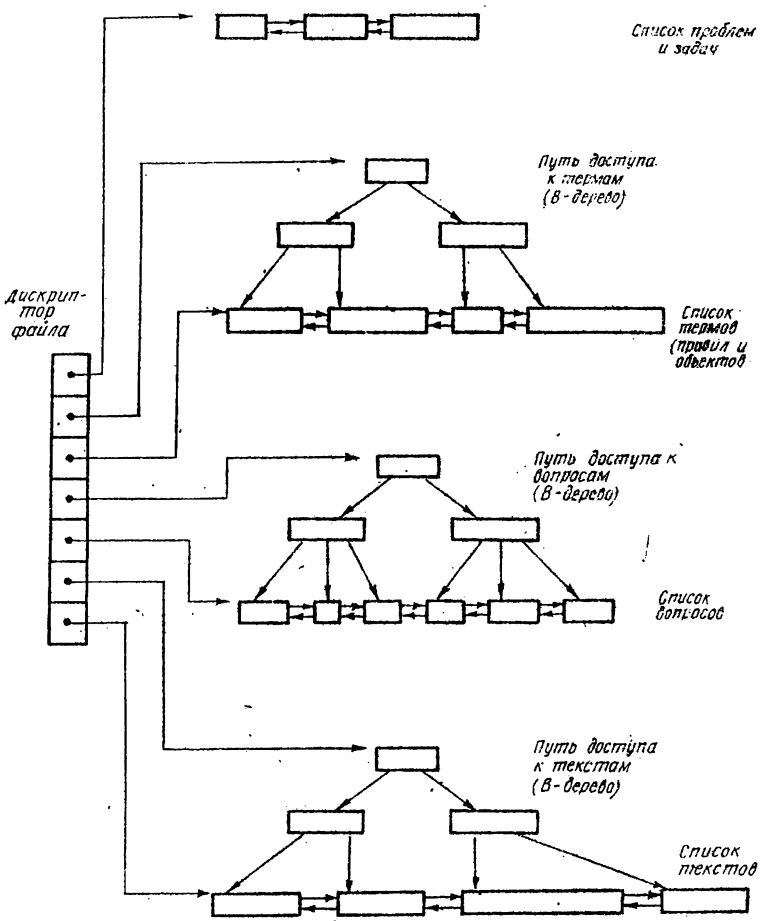



*Рис. 7.6.* Структура файла (модуля) базы знаний

Во-вторых, разделение базы знаний на модули сокращает потребность в оперативной памяти (заметим, время отклика, затраты внешней и внутренней памяти — это главные показатели производительности компьютерных систем [ТИО85]). Модульной базе знаний разрешается иметь огромный размер, т.е. намного превышать объем оперативной памяти ЭВМ. Управляя загрузкой и вытеснением информации, относящейся в оперативной памяти к каждому модулю, можно согласовывать затраты памяти с временем отклика, добиваясь, к примеру, минимального времени отклика при имеющихся в распоряжении ресурсах памяти. Для этого ряд наиболее важных модулей объявляют невытесняемыми и сохраняют в памяти на весь срок решения проблемы, другие модули загружают в память и вытесняют оттуда по оконча-

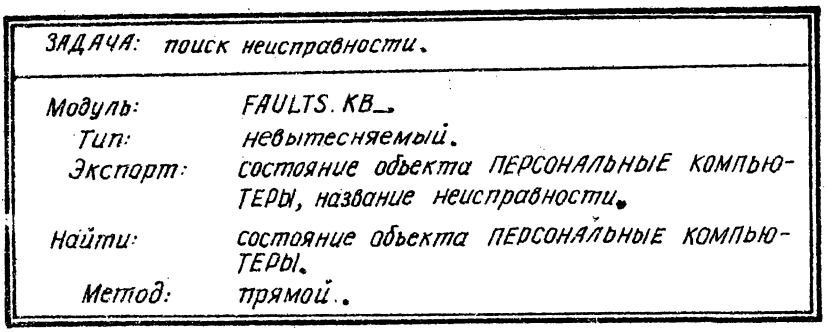



*Рис. 7.7.* Определение модуля в описателе задачи

нии решения соответствующих задач, а третьи — вообще никогда не загружают в память, читая отдельные их элементы непосредственно с внешнего носителя. Кроме того, следует учитывать, что каждый модуль, за исключением корневого, имеет свою собственную локальную рабочую область, которая вытесняется на внешний носитель, когда обработка модуля завершается (позднее она используется для объяснений). В глобальной рабочей области сохраняются лишь результаты работы корневого, загруженного первым модуля, а также те промежуточные и окончательные результаты дедукции для каждого некорневого модуля, которые понадобятся для решения последующих задач. Эти результаты образуют так называемый список экспортируемых значений модуля (рис. 7.7).

Наконец, в-третьих, модульный подход упрощает проектирование базы знаний, так как в каждом модуле содержится относительно обособленный набор элементов, который можно составлять, изменять и отлаживать автономно. Так, любой модуль разрешается помещать в базу знаний отдельно от остальных. Он может быть извлечен из базы знаний, изменен, согласован, а затем помещен обратно и никакие другие модули при этом не затрагиваются. Тем не менее обособленность, независимость модулей относительна, поскольку все они обмениваются фактами через рабочую область.

Допускается множество вариантов разделения базы знаний на модули. Выбор конкретного варианта осуществляет разработчик интеллектуальной системы исходя из специфики предметной области и решаемой проблемы. Прежде всего он может вообще отказаться от выделения модулей, и такая база знаний считается состоящей из единственного модуля, описывать который не нужно. При другом варианте либо базу знаний делят на невытесняемые модули, и тогда они все находятся в оперативной памяти и не удаляются оттуда, либо часть модулей объявляют вытесняемыми, по мере надобности автоматически заменяемыми другими. В обоих последних вариантах необходимо следить за тем, чтобы в памяти помещались все модули, которые должны быть активны одновременно. Применение незагружаемых модулей позволяет никогда не заботиться о ресурсах оперативной памяти и размерах задач, однако эффективность решения проблемы может оказаться невысокой. Часто ее удается существенно повысить, сделав некоторые особо важные или близкие к корневому модули невытесняемыми либо вытесняемыми, но все-таки загружаемыми в оперативную память.

## СПИСОК ЛИТЕРАТУРЫ


[АБИ90] *Абилов В.Г., Зинченко Н.И.* Tyrbo и Arity: два подхода к логическому программированию // Мир ПК. — 1990. — № 2. — С. 32—42; № 3. — С. 31—43.

[БАК92] *Бакаев А.А., Грицвнко В.И., Козлов Д.Н.* Экспертные системы и логическое программирование. — Киев: Наук. думка, 1992. — 218 с.

[БРА90] *Братко И.* Программирование на языке Пролог для искусственного интеллекта. — М. : Мир, 1990. — 560 с.

[ДЕЙ88] *Дейт К.* Руководство по реляционной СУБД DB/2. — М. : Финансы и статистика, 1988. — 320 с.

[ДЕЙ90] *Дейт К.* Введение в системы баз данных. В 2 кн. — М. : Наука, 1989—1990.

[ВУЛ87] *Вулф А.* Универсальный комплекс для разработки экспертных систем реального времени // Электроника. — 1987. — № 7. — С. 43—46.

[КЛА87] *Кларк К., Маккейб Ф.* Введение в логическое программирование на микро-Прологе. — М. : Радио и связь, 1987. — 312 с.

[КОВ90] *Ковальски Р.* Логика для решения проблем. — М. : Наука, 1990. — 280 с.

[ОСУ90] *Приобретение* знаний / Под ред. С.Осуга, Ю.Саэки. — М. : Мир, 1990. — 304 с.

[ТИО85] *Тиори Е., Фрай Дж.* Проектирование структур баз данных. В 2 кн. — М. : Мир, 1985. — 507 с.

[УОТ89] *Уотермен Д.* Руководство по экспертным системам. — М. : Мир, 1989. — 388 с.

[УЭН89] *Представление* и использование знаний / Под ред. Х.Уэно, М.Исидзука. — М. : Мир, 1989. — 220 с.

[ФОР87] *Экспертные* системы. Принципы работы и примеры / Под ред. Р.Форсайта. — М. : Радио и связь, 1987. — 224 с.

[ЭЛТ87] *Элти Дж., Кумбс М.* Экспертные системы: концепции и примеры. — М. : Финансы и статистика, 1987. — 191 с.

[ABA87] *Abarbanel R., Williams M.* A relational representations for knowledge bases // Proc. First intern. conf. on expert database systems / Ed. by L. Kershberg. — Menlo Parc, CA: Benjamin/Cummings, 1987. — P. 191—206.

[ALT85] *Alty J.* Use of expert systems // Comput. Aided Eng. — 1985. — N 2. — P. 2—9.

[ANE82] *A new* knowledge representation for diagnosis in rheumatology / Lindberg, L., Kingsland, L., Roeseler, G., et al. // Proc. of AMIA congress 82. — New York: Masson Publishing Co., 1982. — P. 299—303.

[BAR81] *Barnet J.* Computational methods for a mathematical theory of evidence // Proc. 7th inter. joint conf. artif. intel. — Vancouver, 1981. — P. 868—875.

[BEN85] *Bennet J.* ROGET: A knowledge-based consultant for acquiring the conceptual structure of diagnostic expert system // J. Automated Reasoning. — 1985. — Vol. 1. — P. 49—74.

[BOD89] *Boden M.* AI dangers // Comput. Bull. — 1989. — N 9. — P. 22—23.

[BOB87] *Bobrow D., Mittals S, Stefic M.* Expert systems: perils and promise // Commun. ACM. — 1987. — Vol. 29, N 9. — P. 880—894.

[BUX89] *Buxton R.* Modelling uncertainity in expert systems // Int. J. Man-Mach. Stud. — 1989. — Vpl. 31. — P. 415—476.

[COD82] *Codd E.* A relational database: a practical foundation for productivity // Commun. ACM. — 1982. — Vol. 25, N 2. — P. 123—139.

[DEL88] *Delcambre L., Etheredge J.* The Relational production language: a production language for relational databasees // Proc. Second intern. conf. on expert database systems / Ed. by L.Kerschberg. — Menlo Parc, CA: Benjamin / Cummings, 1988. — P. 153—162.

[ESH86] *Eshelman L., McDermott J.* MOLE: A knowledge acquisition tool that uses its head // Proc. Fifth Nat. conf. artif. intel. — 1986. — P. 950—960.

[FEI89] *Feigenbaum E.* Expert systems expert // Comput. Bull. — 1989. — N 6. — P. 22—23.

[FIK85] *Fikes R., Kehler T.* The role of frame-based representation in reasoning // Commun. ACM. — 1985. — Vol. 28, N 9. — P. 904—920.

[FIK85] *Fink P., Lusth J., Duran J.* A general expert system design for diagnostic problem solving // IEEE Trans. Pattern Analysis and Machine Intel. — 1985. — Vpl. 7, N 5. — P. 553—560.

[FST87] *1stCLASS* Instruction Manual. — Wayland, USA: Programs in Motions Inc., 1987. — 238 p.

[GAL85] *Gallant S.* Connectionist expert systems // Commun. ACM. — 1985. — Vol. 31, N 2. — P. 152—163.

[GAR81] *Garvey T., Lowrance J., Fischler M.* An inference technique for integrating knowledge from disparate sources // Proc. 7th Intern. joint conf. artif. intel. — Vancouver, 1981. — P. 319—325.

[GOR85] *Gordon J., Shortliff E.* A method for managing evidential reasoning in a hierarchical hypotesis space // Artif. Intel. — 1985. — Vol. 26. — P. 323—357.

[GUR85] *GURU* Reference Manual. — Latayette, USA: Micro Database Systems, 1985. — 326 p.

[ISH81] *Ishizuka M., Yao J.T.P.* Inexact inference for rule-based damage assessment of existing structures // Proc. 7th intern. joint conf. artif. intel. — Vancouver, 1981. — P. 838—842.

[HAR85] *Harmon P., King D.* Expert Systems. Artificial Intelligence in Business. — New York: Wiley press, 1985. — 289 p.

[HAY85] *Hayes-Roth F.* Rule-based systems // Commun. ACM. — 1985. — Vol. 28, N 9. — P. 921—941.

[KAH85] *Kahn G., Nowlan S., McDermott J.* Strategies for knowledge acquisition // IEEE Trans. Pattern Analysis and Machine Intel. — 1985. — Vol. 7, N 5. — P. 511—522.

[LAF82] *Lafue G.* Basis decisions about linking an expert systems with a DBMS: a case study // Database Eng. — 1982. — Vol. 2. — P. 238—246.

[LEN82] *Lenat D.* EURISCO: a programm that learns new heuristics and domain concepts // Artif. Intel. — 1982. — Vol. 21, N 1. — P. 61—99.

[LEN90] *Lenat D., Guha R., Pittman K., Pratt D., Shepherd M.* Cyc: towards programs with common sense // Commun. ACM. — 1990. — Vol. 33, N 8. — P. 30—49.

[MAR88] *Mertorelli W.* PC-based expert systems arrive // Datamation. — 1988. — Vol. 34, N 7. — P. 56—66.

[MEA90] *Meador G., Mahler E.* Choosing an expert systems game plan // Ibid. — 1990. — Vol. 36, N 15. — P. 64—69.

[MOA87] *Moadd J.* Building a bridge to expert systems // Ibid. — 1987. — Vol. 33, N 1. — P. 17—19.

[MOL88] *Moldovan, D., Wu, C.I.* A hierarchical knowledge based systems for airplane classification // IEEE Trans. Software Eng. — 1988. — Vol. 14, N 12. — P. 1829—1834.

[PRA85] *Prade D.* A computational approach to approximate and plausible reasoning with applications to expert systems // IEEE Trans. Pattern Analysis and Machine Intell. — 1985. — Vol. 7, N 3. — P. 260—283.

[QUI83] *Quinlan J.* Inferno: a cautious approach to uncertain inference // Comput. J. — 1983. — Vol. 26. — P. 255—269.

[SCH88] *Schoen E., Smith R., Buchanan B.* Design of knowledge-based systems with a knowledge-based assistant // IEEE Trans. Software Eng. — 1988. — Vol. 14, N 12. — P. 1771—1791.

[SHA76] *Shafer G.* A Mathematical Theory of Evidence. — Princeton: Princeton Univ. press, 1976. — 297 p.

[SHO76] *Shortliffe E.* Computer-Based Medical Consultations: MYCIN. — New York: Elsevier, 1976. — 296 p.

[STO87] *Stonebraker M., Hanson E., Potamianos S.* A rule manager for relational database systems. — Berkeley, 1987. — 38 p. — (Memorandum N UCB/ERL M87/38).

[ZAD78] *Zadeh L.* Fuzzy sets as a basis for a theory of possibility // Fuzzy Sets Syst. — 1978. — Vol. 1, N 1. — P. 3—28

## ОГЛАВЛЕНИЕ

**Наукове видання**

Академія наук України

Інститут кібернетики ім. В.М.Глушкова АН України

*БАКАЄВ Олександр Олександрович*

*ГРИЦЕНКО Володимир Ілліч*

*КОЗЛОВ Дмитро Миколайович*

**МЕТОДИ ОРГАНІЗАЦІЇ ТА ОБРОБКИ БАЗ ЗНАНЬ**

Київ "Наукова думка"

*Російською мовою*

Художній редактор *Г.О.Сергєєв*
Технічний редактор *Н.І.Кудрик*
Оператор *В.М.Курган*
Коректор *Г.Т.Коровніченко*

---

Здано до набору 11.02.93. Підп. до друку 30.12.92. Формат 60x84/16. Папір друк. № 2. Гарн. Універс. Друк. офс. Ум. друк. арк. 8,84. Ум. фарбо-відб. 8,84. Обл.-вид. арк. 10,43. Зам. 3-227. Тираж 1000

Оригінал-макет підготовлено у видавництві "Наукова думка". 252601, Київ 4, вул. Терещенківська, 3.
Київська книжкова друкарня наукової книги. 252004 Київ 4, вул. Терещенківська, 4.

**ИЗДАТЕЛЬСТВО "НАУКОВА ДУМКА" В 1993 г. ВЫПУСТИТ В СВЕТ КНИГУ:**

**Кибернетика и вычислительная техника:** Межвед. сб. науч. тр. — Киев: Наук. думка, 1965. — ISSN 0452-9910. **Вып. 98.** Медицинская кибернетика. — 1993. — 10 л.

В сборнике рассматриваются актуальные вопросы биологической и медицинской кибернетики: автоматизация сбора и обработки медицинских данных, алгоритмы машинной диагностики, контроль функционального состояния человека, влияние факторов внешней среды на биологические ритмы, математические модели биологических ритмов.

Для научных работников, инженеров, биологов и медиков, а также аспирантов, студентов.

**Кибернетика и вычислительная техника:** Межвед. сб. науч. тр. — Киев: Наук. думка, 1965. — ISSN 0453-9910. **Вып. 100:** Эргатические системы управления. — 1993. — 10 л.

В сборнике приводятся результаты исследований по разработке теоретических основ анализа и синтеза эргатических систем управления, т.е. систем управления с человеком-оператором в контуре управления. Рассматриваются вопросы синтеза законов, выбора и формализации стратегий управления, построения структур эргатических систем управления. Освещаются вопросы распознавания образов, теорий конфликтов, имитационного моделирования, разработки алгоритмов функционирования эргатических систем управления. Исследуются эргатические системы, предназначенные для работы в промышленности, на воздушном и морском транспорте.

Для специалистов в области кибернетики и эргономики, преподавателей вузов, аспирантов, студентов.

**Кибернетика и вычислительная техника:** Межвед. сб. науч. тр. — Киев: Наук. думка, 1965. — ISSN 0452-9910.
**Вып. 99:** Сложные системы управления. — 1993. — 10 л.

Статьи сборника посвящены разработке и применению дифференциально-геометрических, алгебраических и вычислительных методов в теории управляемых динамических систем. Исследуются вопросы управляемости, устойчивости, стабилизации, оптимизации и идентификации систем с сосредоточенными и распределительными параметрами, описываются методы синтеза и анализа конкретных систем управления различной физической природы.

Рассматриваются математические задачи, связанные с построением баз знаний, искусственным интеллектом и автоматизированными системами принятия решений.

Для научных работников и инженеров, интересующихся теорией автоматического управления и ее приложениями.

*Предварительные заказы на эту книгу принимают местные магазины книготоргов.*

*Просим пользоваться услугами новой торговой сети "Академическая книга" АН Украины, в которую вошли книжные магазины "Наукова думка" (г. Киев, ул. Грушевского, 4), "Академическая книга" (г. Харьков, ул. Чернышевского, 87), "Академическая книга" (г. Днепропетровск, проспект Гагарина, 24), а также книжные магазины Киевского книготоргового объединения "Академическая книга": № 1 (ул. Ленина, 42), № 2 (проспект академика Вернадского, 79), № 3 (проспект "Правды", 80-а), № 4 (ул. Пирогова, 10-в), № 7 (ул. Сретенская, . 7).*