МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОУ ВПО «СИБИРСКАЯ ГОСУДАРСТВЕННАЯ ГЕОДЕЗИЧЕСКАЯ АКАДЕМИЯ»

Ю.А. Кравченко

ОСНОВЫ КОНСТРУИРОВАНИЯ СИСТЕМ ГЕОМОДЕЛИРОВАНИЯ

Книга 1

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАЦИОННОГО ГЕОМОДЕЛИРОВАНИЯ

Часть 2

Новосибирск
СГГА
2008

УДК 528.91
К772

Рецензенты:

Доктор технических наук, профессор
Томского государственного университета
А.В. Скворцов

Кандидат технических наук, доцент Новосибирского
государственного архитектурно-строительного университета
А.Ф. Задорожный

Кравченко, Ю.А.
К 772 Основы конструирования систем геомоделирования. Книга 1. Теоретические основы информационного геомоделирования. Часть 2 [Текст] : монография / Ю.А. Кравченко. – Новосибирск: СГГА, 2008. – 288 с.

ISBN 978-5-87693-299-0 (ч. 2)
ISBN 978-5-87693-297-6 (кн. 1)
ISBN 978-5-87693-296-9

Содержатся сведения, необходимые для понимания структуры и функционирования современных и перспективных систем информационного геомоделирования: общие характеристики систем, элементы теории систем обработки данных и реляционных баз данных, систем искусственного интеллекта.

Для студентов старших курсов, аспирантов и специалистов в области информационного геомоделирования, геоинформатики и картографии.

Печатается по решению редакционно-издательского совета СГГА

Научный редактор: кандидат технических наук,
профессор Сибирской государственной геодезической академии
Ю.Г. Костына

УДК 528.91
ISBN 978-5-87693-299-0 (ч. 2)
ISBN 978-5-87693-297-6 (кн. 1)
ISBN 978-5-87693-296-9

СОДЕРЖАНИЕ

4. МОДЕЛИ И СИСТЕМЫ

4.1. Моделирование

Моделирование стало одним из наиболее распространенных методов человеческой деятельности. В философской и общенаучной литературе под моделированием понимается познание и отображение в теории тех или иных сторон материальной действительности. В конкретных научных дисциплинах *моделированием* называют процесс представления и изучения реальных объектов или процессов с помощью моделей. Поэтому различают моделирование в общепознавательном, гносеологическом смысле и в более утилитарном, узком, специальном. Гносеологические модели выполняют две функции: объяснение и предсказание. Моделирование в гносеологическом смысле не является предметом нашего рассмотрения, поэтому ниже будет говориться о специальном моделировании.

В естественных науках моделью какой-либо системы называют описание этой системы на языке той или иной научной теории в виде формулы, уравнения или системы уравнений, фрагмента теории или всей теории в целом. Примером наиболее известных естественнонаучных теорий являются геоцентрическая система мира Птолемея и гелиоцентрическая модель Н. Коперника.

Центральным понятием при любой трактовке моделирования является понятие модели. В широком смысле слова *модель* – это «образ (в том числе условный или мысленный – изображение, описание, схема, чертеж, график, план, карта и т. п.) или прообраз (″оригинала″ данной модели), используемый при определенных условиях в качестве их ″заместителя″ или ″представителя″» [2, т. 16, с. 399]. Так как модель может быть и образом, и прообразом, то далее под моделью можно понимать материальный или идеальный объект, служащий источником информации об определенных свойствах другого объекта.

Потребность в моделировании возникает в связи со сложностью объектов, невозможностью или экономической нецелесообразностью их непосредственного изучения, риском для человеческой жизни, большими затратами времени и рядом других причин.

В основе моделирования как метода лежит *аналогия*, сходство различных объектов в некоторых выделенных отношениях. Если оригинал и модель обладают рядом одинаковых свойств, то можно ожидать, что другие их характеристики будут также достаточно близкими. Но отсюда следуют и ограничения на возможность моделирования: схожие признаки оригинала и модели должны являться наиболее существенными в выделенном отношении. Кроме того, изучаемые признаки должны быть тесным образом связаны с характеристиками, на основании которых модель была отождествлена с оригиналом. Отождествление модели с оригиналом по второстепенным, случайным признакам является основной причиной неадекватности модели оригиналу.

Известно сопоставление понятий модели и закона. В отличие от модели, один закон не может быть лучше или хуже другого. О законе можно говорить, что он

либо правилен, либо неправилен. Но в то же время о разных моделях можно утверждать, что одни лучше или более адекватны, чем другие.

Перенос свойств модели на оригинал носит вероятностный характер. Вообще же, чем больше мощность множества тождественных признаков модели и оригинала, тем выше вероятность совпадения признаков, переносимых с модели на оригинал. Однако, безусловное стремление к максимальному сходству между моделью и оригиналом считается методологической ошибкой. По мере увеличения адекватности сложность модели приближается к сложности воспроизводимого объекта, и применение самого метода моделирования теряет смысл.

Также ошибочным является проявление максимализма в противоположном направлении – стремление к предельному упрощению моделей: «…если мы преследуем цель упрощения изучаемого объекта при моделировании в каких-либо определенных отношениях, то нет никакого резона требовать, чтобы модель была во всех отношениях проще ″оригинала″ – наоборот, имеет смысл пользоваться любым, сколь угодно сложным арсеналом построения моделей, лишь бы они облегчали решение проблем, ставящихся в данном конкретном случае» [2, т. 16, с. 399]. Следовательно, адекватность и сложность моделей могут и должны быть предметом оптимизации.

Понятие модели может быть уточнено через понятие изоморфизма и гомоморфизма. Как отмечалось выше, *изоморфизмом* называют такое отображение одного множества на другое, когда каждому элементу первого множества определенным образом ставится в соответствие только один элемент второго множества и, наоборот, произвольному элементу второго множества соответствует лишь один элемент первого.

В силу уникальности, неповторимости каждого конкретного реального объекта отношение *строгого изоморфизма* может быть только между абстрактными объектами. Поэтому толкование моделирования невозможно без привлечения идеи *гомоморфизма*, под которым понимают такое отношение между множествами, когда однозначное соответствие соблюдается лишь в одном направлении, но не выполняется в другом.

Тогда отношение между моделью и оригиналом может рассматриваться как отношение изоморфизма между гомоморфным образом оригинала и гомоморфным образом модели. Из такой трактовки моделей вытекают такие их свойства, как:

– *рефлексивность* – объект является моделью самого себя;

– *симметричность* – если объект *A* является моделью объекта *B*, то и *B* может служить моделью *A*;

– *транзитивность* – если *A* является моделью *B*, а *B* есть модель объекта *C*, то *A* является моделью *C*.

Отношение, обладающее свойствами рефлексивности, симметричности и транзитивности, называют *отношением эквивалентности*. Поэтому считают, что моделирование является отношением эквивалентности между двумя системами. Свойство симметричности моделей подтверждает принятое выше

определение модели. В силу свойства симметричности нет необходимости трактовать модель как образ или как прообраз. Это излишне, так как при определенных условиях оригинал может рассматриваться как модель модели. Таким образом, модели можно рассматривать просто как некоторый класс объектов.

По характеру воспроизводимых сторон объекта модели подразделяют на *структурные* и *функциональные*. В зависимости от природы используемых моделей, различают предметное («физическое», материальное или экспериментальное) и знаковое (идеальное или теоретическое) моделирование. В процессе *предметного моделирования* в качестве моделей используются материальные объекты, и в моделях воспроизводятся основные геометрические, физические, динамические, функциональные и другие характеристики оригиналов. При *знаковом моделировании* моделями служат знаковые образования какого-либо вида: схемы, чертежи, карты, графы, формулы, выражения некоторого языка. По этому же признаку *модели* делят соответственно на *предметные* и *знаковые*. Знаковые последовательности и их элементы обычно рассматриваются вместе с определенными преобразованиями или операциями, которые могут над ними выполняться человеком или техническим средством.

Наиболее полно информационный характер моделирования проявляется в процессе создания и обращения знаковых моделей. При знаковом моделировании могут использоваться как самые общие представления, так и хорошо разработанные знаковые системы. В первом случае говорят о *содержательном моделировании*, во втором – о *формальном* (часто называемым *математическим*). Необходимо, однако, указать на определенную условность разделения моделирования на содержательное и формальное. Так или иначе, любое знаковое моделирование неотделимо от абстракции и идеализации. И описание реальных объектов в содержательных терминах есть также абстрактное описание. Особенность *содержательного моделирования* проявляется в том, что оно осуществляется в терминах и понятиях конкретной научной дисциплины. Используемые содержательные понятия по необходимости имеют смысл и значение. В процессе неформального описания проблемной области специалист проецирует понятия на обозначаемые ими объекты реального мира, их свойства и отношения.

Другой особенностью содержательного моделирования является нечеткость, многозначность, размытость используемых терминов. Понятия могут приобретать множество оттенков в зависимости от контекста. Различные индивиды неизбежно трактуют одно и то же понятие в соответствии с собственным практическим и языковым опытом. Кроме того, сами понятия имеют свойство изменяться по мере развития конкретной теоретической системы.

Наконец, еще одним свойством неформальных (естественных или этнических) языков служит то обстоятельство, что в них отсутствуют однозначные правила оперирования понятиями. Терминология прикладной

дисциплины может рассматриваться как подмножество неформального языка.

Формализмом называют некоторое исчисление, позволяющее заменить операции с объектами операциями со знаками, поставленными в соответствие объектам. Термины «формализм» и «исчисление» обычно употребляются как синонимы. В формальных системах все операции над символами осуществляются при полном абстрагировании от содержания, обозначаемого этими символами. Формулы или формальные выражения представляют собой формальные образы соответствующих высказываний об объектах предметной области. При этом предполагается, что истинность формальной системы может быть установлена только при интерпретации (подтверждении) ее в терминах какой-либо содержательной теории.

Все технические дисциплины являются в той или иной мере формализованными теориями, то есть теориями, изучающими объекты с помощью операций над знаками, совершающихся по правилам, которые определяются только формой принятых в данной теории знаков, представляющих объекты и их связи. В отличие от неформализованных (интуитивных) теорий, в формализованной теории свойства элементарных понятий задаются точным аксиоматическим методом.

Формализованный язык отличается от обычного тем, что является системой таких знаков, операции с которыми совершаются по правилам, которые определяются только формой выражений, составленных из конечного числа символов. Если в обычных языках многозначность – обычное явление, то при создании формализованных языков стремятся к полной однозначности и предельной точности символов. При этом однозначность провозглашается как логический принцип, согласно которому каждая грамматическая функция выражается только одним знаком и каждый знак выражает только одну функцию.

Всякий формализованный язык имеет свой синтаксис – систему правил, определяющих структуру, построение и преобразования синтаксически осмысленных выражений формализованного языка. Таким же образом любой формализованный язык содержит и семантику, которая изучает значение выражений языка, соотношение между формальной системой и ее интерпретацией, проблемы истинности – соотношения между символами и обозначаемыми с их помощью объектами предметной области.

Обычный язык не пригоден для описания проблем, где требуется безупречная точность формулировок. Но описание формализованного языка осуществляется с помощью обычного языка, который играет при этом роль метаязыка.

К преимуществам *формальных языков* относят однозначность, точность, лаконичность и ясность. Однако последнее качество, приписываемое формальным языкам, видимо, имеет место не всегда. В качестве иллюстрации к этому утверждению можно привести формулировку теоремы Гольбаха на естественном языке: «Каждое четное натуральное число представимо в виде суммы двух простых чисел» и ее формальную запись:

$$\forall x\{\, 2 \mid x \rightarrow \exists y \exists z[\, y + z = x \wedge (\,\mathrm{Prim}\, y \wedge \mathrm{Prim}\, z\,)]\},$$

где $\forall$ – квантор общности;

$\exists$ – квантор существования;

$\rightarrow$ – знак логической импликации.

Наиболее существенным отличием формального языка от естественного служит то, что наряду с *алфавитом* он содержит точные *правила оперирования знаками*. Обязательными элементами формального языка являются:

- конечное множество исходных символов;
- ограниченное число правил образования конечных последовательностей символов (формул, выражений);
- некоторое множество правил вывода одних формул из других.

Основное назначение формального языка – замещение реальных объектов, их свойств или отношений последовательностями символов. Совокупность правил оперирования знаковыми выражениями образует некоторое *исчисление* или *формализм*. *Формальная система* образуется, если исчисление дополняется некоторой *системой аксиом* – конечным множеством исходных выражений формального языка. Все остальные выражения формального языка выводятся из исходных с помощью формальных правил.

Система аксиом должна быть непротиворечивой, полной и независимой. *Непротиворечивой* называется такая система аксиом, в которой ни одно исходное выражение не может быть ни доказано, ни опровергнуто. *Полнота* системы аксиом означает, что все положения формальной теории могут быть получены из исходных выражений. *Независимость* аксиом предполагает, что ни одна из них не может быть получена из остальных аксиом.

Из результатов, полученных К. Геделем, следует, что любой формализации присуща внутренняя ограниченность: средствами формальной теории не может быть доказана ни ее ложность, ни ее истинность. Поэтому разработка формальной системы должна включать в себя и ее интерпретацию. Под *интерпретацией* понимается истолкование положений формальной теории в терминах хотя бы одной содержательной теории. Формальная теория считается *истинной*, если ее высказывания соответствуют утверждениям некоторой неформальной теории при условии, что последняя сама непротиворечива.

Самым замечательным свойством формальной системы является возможность оперирования знаковыми выражениями, полностью абстрагируясь от содержания соответствующей неформальной теории. Это свойство позволяет получать утверждения формальной системы с помощью технических средств переработки информации, используя только аксиомы и правила формального вывода. Технические средства переработки информации – это суть аналоговые

(непрерывные) и цифровые (дискретные) электронные вычислительные машины. Аналоговые ЭВМ по сравнению с дискретными получили столь незначительное распространение, что в настоящее время, когда говорят об электронных цифровых вычислительных машинах, слово «цифровые» обычно опускают. Моделирование же с помощью дискретных ЭВМ принято называть *цифровым*. Происхождение термина «цифровое» объясняется тем, что представление символов в ЭВМ может трактоваться как последовательность двоичных цифр.

Таким образом, *цифровые модели* понимаются как математические модели, представленные на языке ЭВМ. Поэтому имеет смысл рассмотреть более детально свойства математических моделей.

4.2. Математические модели географического пространства

В [6] отмечается, что математическая теория никогда не исследует свойства природных объектов как таковых; объектом ее исследования могут быть лишь абстрактные объекты. В [14, с. 330] математическая модель трактуется как «класс неопределяемых (абстрактных, символических) математических объектов … и отношения между этими объектами». Аналогичные определения математических моделей с теоретико-множественных позиций стали традиционными.

Ю.А. Шрейдер и А.А. Шаров пишут: «В математике моделью (или реляционной системой) называется некоторое множество *M* с заданным на нем набором отношений $\{r_1, r_2, ..., r_n\}$« [26, с. 22]. В работе [16, с. 11] вместо понятия модели используется понятие «структура»: «Математическими структурами называют множества (или их совокупности), между элементами которых установлены отношения».

В последнем определении можно было бы увидеть противоречие двум предыдущим, но на самом деле его нет, так как в математике модели обычно рассматриваются с точностью до изоморфизма, т. е. не конкретные модели, а их классы. Структура (совокупность отношений) есть то общее, что позволяет выделить данный класс из всех математических моделей. Поэтому понятия «структура» и «модель с данной структурой» математики иногда склонны отождествлять, так как с их точки зрения не важно, какова природа элементов, между которыми выполняются отношения. Определение структуры уже предполагает наличие тех или иных элементов. «Но при описании реальных объектов приходится различать модель как множество элементов определенной природы, связанных присущими модели отношениями, и структуру как абстрактную категорию» [26, с. 26].

Хотя в математике описываются некоторые объекты, их свойства и отношения, сами объекты при этом не указываются. Вследствие этого в определении математической модели отсутствует зависимый родительный падеж (модель чего?) [26]. Прикладной специалист, в отличие от математика, обязан за абстрактными категориями видеть реальные объекты, свойства и отношения. В естественнонаучных и технических дисциплинах формальные понятия неизбежно подлежат интерпретации.

Давая определение математической модели географического пространства, будем исходить из общего определения математических моделей. Поэтому *математическую модель A географического пространства G* будем понимать как некоторое *базовое множество P*, элементы которого интерпретируются как объекты географического пространства, и *структуру S* – совокупность заданных на нем отношений, то есть

$$A = \{P, S\}.$$

Под географическим пространством, или геопространством, здесь понимается земная поверхность в целом или ее некоторая часть вместе с находящейся на (над, под) ней совокупностью выделенных реальных или мыслимых (виртуальных) объектов и явлений. Таким образом, это всегда будет какой-то фрагмент географического пространства, поскольку моделирование всего геопространства – неосуществимая задача даже в будущем как в силу ограниченности ресурсов человечества, так и по причине конкретности решаемых задач. Объектами математического моделирования могут быть отдельные геосистемы, их фрагменты и/или объединения некоторого числа геосистем.

Под *методом математического моделирования* географического пространства будем понимать тройку

$$M = (Sem, A, F),$$

где Sem – содержательная модель объекта или модель объекта в содержательных терминах;

A – его математическая модель;

$F : Sem \to A$ – отображение содержательной модели в математическую.

Иными словами, метод математического моделирования геопространства представляет собой совокупность объектов содержательной теории (предметной области), математической модели и некоторого отображения (формализации), ставящего в соответствие абстрактным объектам предметной области *Sem* элементы из базового множества модели *A* и отношениям из *Sem* – отношения в *A*. Отображение *F* не обязательно является изоморфизмом или хотя бы гомоморфизмом. В общем случае оно может быть сколь угодно сложным.

Решение задачи моделирования геопространства не является однозначным. Для представления одной и той же выделенной совокупности объектов геопространства могут использоваться различные математические модели. При их выборе руководствуются соображениями удобства и эффективности.

Обратное отображение $I = F^{-1} : A \to Sem$ будем называть *интерпретацией* математической модели географического пространства.

Обычно под математическим моделированием понимают *математическую экспликацию* содержательной модели, формализацию решаемой проблемы. Приведенную формулировку можно рассматривать как частный случай определения математического моделирования в широком смысле.

4.3. Информационные модели географического пространства

Формализация теории дает возможность абстрагироваться от конкретного содержания проблемной области и позволяет, таким образом, использовать технические средства переработки информации. Но реализация математических моделей или методов моделирования на ЭВМ осложняется тем обстоятельством, что последние имеют собственную структуру, отличную от структуры представляемых объектов. Чтобы содержательно говорить о любом информационном моделировании, необходимо рассмотреть вычислительные структуры.

Анализ вычислительных структур может осуществляться на *абстрактном* (логическом) или *конкретном* (физическом) уровне и в *статическом* или *динамическом аспекте* [25]. На абстрактном уровне статической компонентой решаемой проблемы является *структура информации*, а динамической – *структура алгоритма*. На конкретном уровне статической компоненте соответствует *структура памяти* ЭВМ, а динамической – *структура управления* (*система команд*) ЭВМ (табл. 4.1). В процессе решения задачи на ЭВМ структура информации налагается (погружается) на структуру памяти, а структура алгоритма реализуется через структуру управления. «Различия, которые мы выделили выше, имеют решающее значение, если мы хотим, чтобы наши вычисления были хоть сколько-нибудь осмысленными» [25, с. 12].

Таблица 4.1. Компоненты вычислительных структур

Уровень	Аспект	
	Статический	Динамический
Абстрактный	Структура информации	Структура алгоритма
Конкретный	Структура памяти	Структура управления

Целесообразность выделения абстрактного и конкретного уровней подтверждается и с системно-теоретических позиций. Система может мыслиться как *совокупность функций* (действий) [5]. Каждый элемент системы отличается от других элементов своей *ролью*, или *назначением*. Совокупность функциональных характеристик элемента, ради которых он включается в систему, называют *функциональным местом* элемента. Совокупность функциональных мест и отношений между ними образует *функциональную структуру*, или *организацию системы*. Отображение функциональной структуры на реальные элементы и связи между ними называют *реализацией*. Говорят также, что в процессе реализации происходит *наполнение* функциональных мест или их *погружение* на реальные элементы. Разные системы могут быть совершенно идентичны в функциональном отношении и иметь существенно различное исполнение. Это означает, что «организация может быть реализована различными структурами» [5, с. 27]. Так, ЭВМ в своем развитии прошли реализацию на реле, электронных лампах, полупроводниках, на интегральных и сверхбольших интегральных схемах.

Теперь мы имеем все необходимое для определения информационной модели географического пространства. Математическая модель географического пространства – это абстракция, заданная «ни на чем».

Рассматривая вопрос о ее реализации, мы приходим к понятию информационной модели геопространства. Далее *информационную модель географического пространства*, или *геоинформационную модель*, будем понимать как отображение его математической модели на модель ЭВМ или тройку $D = (A, C, U)$, где A – математическая модель геопространства; C – абстрактное представление ЭВМ; U – отображение $U : A \to C$. Иными словами, информационную модель далее будем понимать как реализацию математической модели на конкретном классе ЭВМ.

Таким образом, в процессе информационного моделирования следует выделять три уровня:

1) *содержательный*, на котором описание предметной области осуществляется с применением естественного языка и специальной терминологии;

2) *формальный*, на котором проблемная область представляется с помощью тех или иных математических объектов;

3) *информационный*, когда математическая модель проблемной области представляется в виде, пригодном для обработки техническими средствами.

4.4. Модель ЭВМ

В приведенном выше определении геоинформационной модели появляется такой объект, как модель ЭВМ. Естественным образом может возникнуть вопрос, почему речь идет об отображении на модель ЭВМ, а не на ЭВМ. Ответ состоит в следующем.

Хотя в цитировавшейся выше работе [25] говорилось о конкретном уровне, ее автор, по существу, имел в виду абстрактное представление ЭВМ. Сочетание структуры памяти и структуры управления соответствует не физически конкретной ЭВМ, а целому классу электронных вычислительных машин с полностью эквивалентными структурами. Более того, любая информационная модель, реализованная на ЭВМ C_1 со структурой памяти M_1 и структурой управления L_1, является одновременно реализацией на ЭВМ C_2 при условии, что $M_1 \subseteq M_2$ и $L_1 \subseteq L_2$. Отсюда следует, что при разработке информационных моделей необходимо стремиться к минимуму используемых средств ЭВМ, если требуется обеспечить их (моделей) транспортабельность, переносимость с одного типа ЭВМ на другой. Но, с другой стороны, если мы хотим добиться эффективности создаваемых моделей, то должны использовать по мере необходимости все доступные функции и возможности технических средств. Таким образом, критерии переносимости и эффективности моделей и методов моделирования противоречивы.

Но даже если представить разработку программного обеспечения для ЭВМ, существующей в единственном экземпляре и с уникальной структурой (такие прецеденты имели место), то необходимо отдавать себе отчет в том, что разработчики программного обеспечения имеют дело опять же не с самой машиной, а с ее идеализированным представлением. Поэтому высказывания о конкретном уровне следует понимать в том смысле, что структура информации

и структура алгоритма реализуются через абстрактную структуру определенного класса ЭВМ, выбранного из множества абстрактных представлений всех существующих и/или мыслимых машин. Однако, в работе [25], о которой идет речь, этот нюанс оказался несколько завуалированным.

То же самое можно сказать и о [5], где *реализация* трактуется как погружение функциональных мест на определенный материал, их превращение в (материальные) элементы системы. Но затем утверждается, что «… функциональная структура реализуется сначала в виде монтажной схемы, а затем в процессе изготовления на производстве» [5, с. 27]. То есть, опять-таки, между функциональной структурой системы и ее материальным воплощением располагается абстрактное представление (монтажная схема) субстанциональных компонентов системы.

Структура данных и структура алгоритма могут рассматриваться как своеобразная «монтажная схема» программного обеспечения. Таким образом, мы имеем полное право считать, что отображение математической модели осуществляется на *формальное представление* электронной вычислительной машины. Поэтому ЭВМ C можно рассматривать как сочетание структуры памяти M и структуры управления L, то есть $C = (M, L)$.

Здесь можно вернуться к экспликации понятия реализации. Реализация функциональной структуры представляется как композиция двух отображений. Первое отображение устанавливает соответствие между функциональной структурой и абстрактным представлением системы как совокупности субстанциональных элементов. Второе отображение устанавливает множество соответствий между абстрактными и реальными элементами системы. Рассуждения о реализации функциональной структуры на некоторых материальных элементах следует понимать только в таком смысле.

4.5. Статические вычислительные структуры

В первом приближении структура ЭВМ рассматривается как совокупность структуры памяти и структуры управления. Память ЭВМ представляется как множество некоторых *единиц памяти*, характеризующихся своим *адресом* и *содержимым*. Отсюда вытекают два способа обращения к данным: по их месту (адресу) и по содержанию. *Доступ по адресу* предполагает упорядоченность единиц памяти (рис. 4.1). *Обращение по содержанию* означает идентификацию данных с помощью некоторой части данных, называемой *ключом*, и также требует упорядочения единиц памяти. Упорядочение, или *адресация, памяти* – это отображение множества единиц памяти на множество натуральных чисел $A: M \to \{1, 2, ..., |M|\}$, где $|M|$ – мощность множества M – интерпретируется как *объем памяти* (рис. 4.1).

1	2		|M|

Рис. 4.1. Структура памяти

Адресация памяти. При решении различных задач часто требуется рассматривать группу соседних единиц памяти как единое целое, эту группу называют *квантом памяти*. *Адресом кванта* считается адрес первой

(наименьшей) единицы памяти, входящей в квант. Кванты памяти служат строительным материалом для образования других, более сложных конструкций в памяти ЭВМ. Самый простой и естественный способ получения совокупности квантов памяти – объединение соседних квантов (рис. 4.2). Последовательность соседних квантов памяти называют *вектором памяти*, или *физической структурой памяти*. Вектор памяти – это та физическая структура, которая используется для реализации любой логической структуры данных. Тогда проблема представления данных в памяти ЭВМ сводится к выбору эффективной *адресной функции* – отображения структуры информации (логической компоненты) на структуру памяти (физическую компоненту). Адресная функция может быть реализована с использованием либо последовательного, либо связного метода распределения памяти.

Рис. 4.2. Квант и вектор памяти

При *последовательном распределении* данные располагаются в строго последовательных квантах памяти и могут быть упорядочены одномерно или многомерно. Отображение *r*-мерного упорядоченного множества $(r \in N)$ на множество *N* положительных целых чисел $\sigma : N^r \to N$ называют *структурной функцией*. Отношение $(n_1, ..., n_r)$ называют *индексным вектором*, его компоненты $n_1, ..., n_r$ – *координатами*, а число компонент r – *рангом* (иногда – *размерностью*). Если для каждой компоненты n_i структурной функции σ указать диапазон допустимых значений, то адресная функция $\alpha : N^r \to M$ будет отображением индексного вектора на вектор памяти *M*. При подстановке в индексный вектор конкретных значений координат адресная функция имеет своим значением адрес памяти, где располагается указанный элемент.

Примером последовательного распределения данных может служить размещение матрицы A_{mn} на рис. 4.3, на котором показано, что ее строки и элементы в каждой строке располагаются последовательно. Адресная функция для матрицы A_{mn} в данном случае будет иметь вид:

$$A(a_{ij}) = A_0 + n(i-1) + (j-1),$$

где $A(a_{ij})$ – адрес элемента a_{ij};

A_0 – начальный адрес размещения матрицы.

Приведенная формула является примером построчного размещения двумерного массива в памяти. Если используется размещение элементов двумерного массива по столбцам, то адресная функция имеет вид

$$A(a_{ij}) = A_0 + m(j-1) + (i-1).$$

$$\begin{array}{cccc} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{array}$$

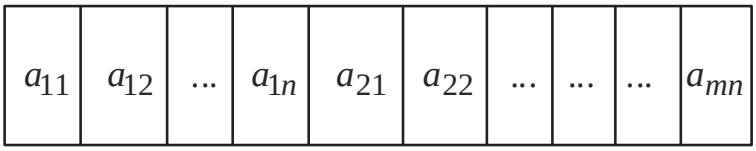


Рис. 4.3. Размещение матрицы

Таким образом, соответствие между структурой информации и структурой памяти устанавливается при помощи двух отображений. Первое отображение устанавливает взаимное однозначное соответствие между элементами данных и конечной последовательностью целых положительных чисел. Второе отображение определяет двухместное отношение между конечной последовательностью целых чисел и совокупностью квантов памяти. Устанавливая определенные отношения между элементами данных, мы тем самым определяем отношения между целыми числами и, как следствие, между квантами памяти.

При *связанном представлении* данных используются указатели. Содержимое квантов памяти может интерпретироваться самым различным образом: как символы, целые числа, числа с плавающей точкой (вещественные числа), адреса, команды. Если содержимое некоторого кванта памяти интерпретируется как адрес, то содержимое такого кванта называют *указателем*. Использование указателей позволяет организовать последовательный доступ к группам квантов, которые могут располагаться в памяти произвольным образом. Начальный адрес последующей и, возможно, предшествующей группы задается в каждой группе с помощью указателя. Связанное представление данных (рис. 4.4) требует большего объема памяти и несколько большего времени при их просмотре, но дает возможность изменять порядок следования, исключать или добавлять данные без их перемещения в памяти.

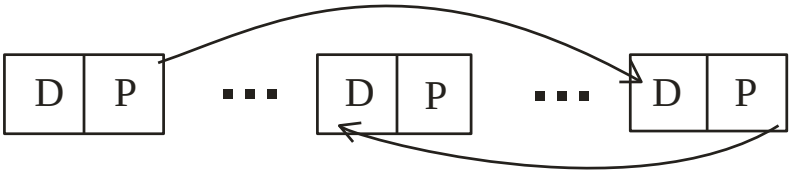


Рис. 4.4. Связанное распределение:
D – данные; P – указатель

Связанное представление наиболее часто используется при управляемом распределении памяти. Вообще же различают три вида распределения памяти: статическое, динамическое и управляемое. При *статическом распределении* необходимая память выделяется программе в момент ее загрузки и освобождается в момент завершения программы. При *динамическом распределении* нужный объем памяти выделяется каждый раз в момент входа в блок программы и освобождается при выходе из блока. При *управляемом распределении* память запрашивается и освобождается программой по мере необходимости. Каждый способ распределения памяти имеет свои преимущества и недостатки, и вопрос о его выборе решается в процессе разработки программы.

Структура информационных моделей, как и математических, представляет собой совокупность отношений между их элементами. Но эти связи в силу особенностей физической организации памяти ЭВМ реализуются через единственное отношение, существующее между квантами памяти компьютера, – отношение порядка (следования, адресации).

В ЭВМ с байтовой организацией основными единицами памяти (рис. 4.5) являются *байт* – 8 двоичных разрядов, или *битов, слово* – следующие друг за другом два байта и *двойное слово* – соседние четыре байта. В некоторых типах ЭВМ два байта называют полусловом, а четыре байта – словом. Нумерацию двоичных разрядов в этих единицах памяти принято осуществлять справа налево (рис. 4.5).

Рис. 4.5. Байтовая организация памяти

Содержимое памяти. Содержимое памяти ЭВМ трактуется как данные, характеризующиеся своим типом и длиной. *Тип данных* определяется способом представления в памяти и совокупностью допустимых операций над данными. В первую очередь данные подразделяются на базовые типы данных и абстрактные типы данных. *Базовые типы данных* – это данные, непосредственно обрабатываемые ЭВМ; к ним относятся данные *арифметического* и *неарифметического типов*. К данным арифметического типа относят целые и вещественные числа, а к данным неарифметических типов – символы и логические данные.

Данные арифметического типа характеризуются основанием системы счисления, способом представления и разрядностью. По способу представления арифметические данные подразделяются на *целые* и *вещественные числа*. Разрядность определяется числом двоичных разрядов, используемых для представления чисел. Если для представления целых чисел используются два байта, то такие целые числа называются *короткими*. Если для представления целых чисел используются четыре байта, то их называют *длинными целыми*.

В представлении целых чисел со знаком старший бит (15-й или 31-й) отводится под знак числа. В представлении вещественных чисел с обычной точностью (4 байта) самый старший разряд используется для знака числа, следующие 7 старших разрядов – для двоичной экспоненты, последние 24 бита (три байта) – для мантиссы числа. В представлении вещественных чисел с удвоенной точностью самый старший бит отводится для знака числа, следующие 11 битов – для двоичной экспоненты числа и младшие 52 разряда – для мантиссы числа. Таким образом, вещественные числа обычной точности могут содержать до 7 значащих десятичных цифр, а вещественные числа удвоенной точности – до 15 десятичных цифр. Диапазоны допустимых значений данных арифметического типа представлены в табл. 4.2.

Таблица 4.2. Представление данных базовых типов

Тип данных	Длина	Область значений
символ	1 байт	256 символов
логические данные	1 байт	0 и 1
короткое целое	2 байта	от -32 768 до 32 767
длинное целое	4 байта	от -2 147 483 648 до 2 147 483 647
короткое целое без знака	2 байта	от 0 до 65 535
длинное целое без знака	4 байта	от 0 до 4 294 967 295
вещественное число	4 байта	от $3{,}4\times10^{-38}$ до $3{,}4\times10^{38}$
длинное вещественное	8 байт	от $1{,}7\times10^{-308}$ до $1{,}7\times10^{308}$

Поскольку для представления символов используется один байт, то отсюда следует, что непосредственно можно представить всего 2^8, то есть 256 различных символов. Логические данные интерпретируются как «ложь», если значение всех битов равно 0; если хотя бы один бит не равен 0, то значение байта интерпретируется как «истина».

Абстрактные типы данных определяются программистом. Они реализуются через базовые типы данных, но для выполнения операций над каждым конкретным абстрактным типом должен быть разработан соответствующий набор функций. Примерами абстрактных типов данных могут служить комплексные числа, даты, единицы измерения углов (градусы, минуты, секунды), английская денежная система и т. п.

Наиболее часто используются такие логические конструкции данных, как массив, структура, линейный список, двунаправленный список, стек, очередь и дек.

Массивы представляют собой последовательности данных, называемых *элементами массива* и имеющих один и тот же тип данных и смысл. Массивы

могут быть одномерными или многомерными. Для указания на конкретный элемент массива используются индексы. Примером одномерного массива является среднемесячная температура в течение года, а примерами двумерных – таблица расстояний между городами или матрица коэффициентов системы линейных уравнений.

Структура данных является последовательностью данных, имеющих различный смысл, даже если типы некоторых ее *элементов* совпадают. Примером структуры могут быть анкетные данные о человеке:

- фамилия, имя, отчество;
- идентификационный номер налогоплательщика (ИНН);
- дата рождения (число, месяц, год);
- место рождения (область, город).

В реальной жизни анкетные данные о сотрудниках содержат массу других сведений, но в демонстрационных целях достаточно и приведенных. В данном примере элементы структуры являются целыми числами и строками (массивами) символов. Такие элементы, как дата рождения и место рождения, сами являются структурами.

Таким образом, элементами структуры могут быть массивы; в свою очередь, элементами массивов могут быть структуры. Так, сведения о сотрудниках предприятия могут быть представлены в виде массива, каждый элемент которого является описанной выше структурой. Для реализации массивов и структур используются вектора памяти.

Линейный список представляет собой структуру данных, содержащую как некоторые данные, так и указатель на следующую аналогичную структуру (см. рис. 4.4). Если продолжить предыдущий пример, то сведения о сотрудниках предприятия могут быть представлены в виде линейного списка.

Очевидно, что данные о сотрудниках желательно представлять в алфавитном порядке вне зависимости от того, как они организованы: как массив или как линейный список. Поскольку в любом предприятии одни сотрудники увольняются, а другие принимаются на работу, при использовании массива придется перемещать данные почти при каждом увольнении и приеме на работу. При использовании линейного списка достаточно будет очевидного изменения значений указателей.

Хотя линейные списки обладают указанным преимуществом перед массивами данных, но и они не лишены того недостатка, что в них можно переходить от одного элемента к другому только в одном направлении в соответствии со значением указателей. Элементы же массива можно просматривать в любом порядке.

Двунаправленные списки представляют собой структуру, содержащую, наряду с данными, указатели на следующий и предыдущий элементы списка (рис. 4.6). Такая организация памяти обеспечивает возможность просмотра элементов списка в прямом и обратном направлениях. Если прямой указатель последнего элемента указывает на первый элемент списка, а обратный указатель первого элемента – на его последний элемент, то такую структуру

называют *двунаправленной кольцевой.* В любой момент времени один из элементов двунаправленного кольцевого списка является первым.

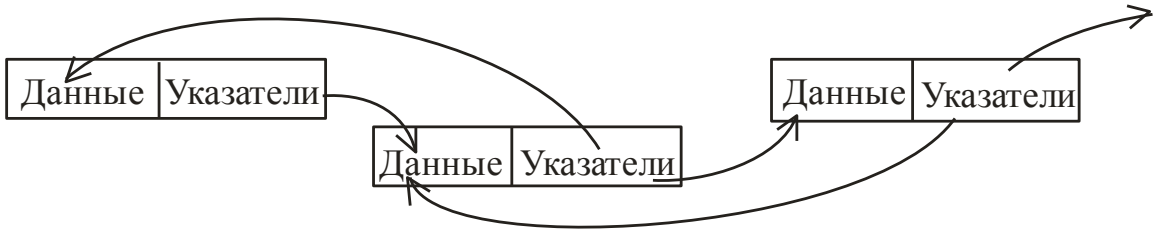


Рис. 4.6. Двунаправленный список

Стеком, или *магазином*, называют структуру, единственным доступным элементом которой в каждый момент времени является элемент, находящийся в *вершине стека* (рис. 4.7). Со стеком возможны всего две операции: добавление и исключение элементов. Доступ к данным стека осуществляется по принципу «последним пришел – первым ушел». Название «стек» произошло от магазина (рожка) автомата, из которого патроны извлекаются в обратном порядке.

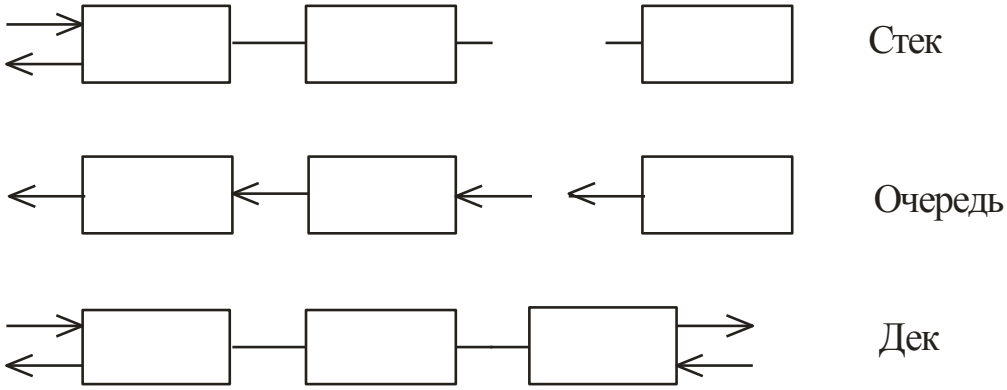


Рис. 4.7. Логические структуры данных

Очередь – последовательная структура данных, в которой добавляемый элемент размещается в ее *конце*, а извлекаются элементы из ее *начала.* Таким образом, доступ к элементам очереди осуществляется по принципу «первым пришел – первым ушел» (см. рис. 4.7).

Дек представляет собой структуру, добавление элементов к которой и их извлечение может осуществляться с двух концов (см. рис. 4.7).

4.6. Динамические вычислительные структуры

На конкретном уровне рассмотрения динамические вычислительные структуры представляют собой систему команд конкретного класса ЭВМ. Все множество машинных команд можно разделить на команды обработки данных и команды управления. *Команды обработки данных* выполняются последовательно одна за другой, и в результате выполнения каждой такой команды происходит изменение содержимого тех или иных единиц памяти и вырабатывается признак результата (0 или 1) выполнения команды.

Команды управления изменяют последовательность выполнения команд и подразделяются на команды безусловного перехода, команды условного перехода и команды перехода с возвратом.

Если в результате последовательного выполнения команд достигается *команда безусловного перехода*, то следующей после нее всегда выполняется команда, адрес которой указан в команде безусловного перехода в качестве операнда. *Команда условного перехода* в зависимости от признака выполнения предыдущей команды передает управление либо следующей команде, либо команде, адрес которой указан в качестве операнда в команде условного перехода. При достижении *команды перехода с возвратом* выполняется последовательность команд, первый адрес которой указан в команде перехода, после чего управление передается команде, следующей за командой перехода с возвратом. Команды управления являются принципиальным отличием ЭВМ от механических вычислительных устройств.

Следует отметить, что машинная арифметика отличается от арифметики, используемой человеком, и является лишь некоторым приближением к последней. По конструктивным соображениям необходимо, чтобы ЭВМ обрабатывали данные определенного формата и фиксированной длины. В «обычной» арифметике таких ограничений нет. Поэтому для решения задачи на ЭВМ требуется:

– представить данные как последовательности символов нужного формата;

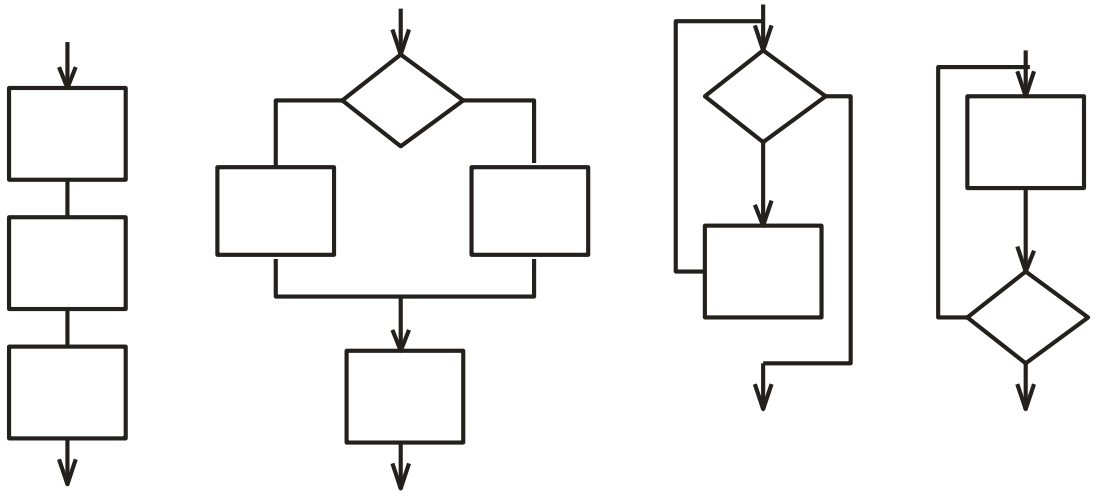
– определить правила интерпретации выходных последовательностей символов (для понимания полученных машиной результатов);

– описать с помощью последовательностей символов характер преобразования (программу) входных последовательностей символов в выходные последовательности. Таким образом, программа представляется как последовательность некоторых элементарных машинных операций, имитирующая выполнение человеком тех или иных действий над данными.

Как только элементарные машинные операции определены, можно создавать последовательности таких операций, необходимые для представления операций более высокого уровня. Так, операции целочисленного умножения и деления могут быть построены на основе операций сложения и вычитания двух целых чисел. Операции с вещественными числами могут быть построены на основе операций с целыми числами. Затем можно определить последовательности операций для вычисления, например, квадратного корня, логарифмов, тригонометрических функций. Далее можно определить операции с комплексными числами или определить операции с матрицами и т. п. Подобное синтезирование последовательностей элементарных операций с целью получения операций все более высокого уровня может быть продолжено до тех пор, пока мы не достигнем возможностей, предоставляемых языками высокого уровня. В конечном итоге можно создать такие последовательности элементарных операций, которые будут соответствовать операциям той или иной проблемной области. Тогда при решении задач на ЭВМ задачи и данные могут представляться в виде, естественном для конкретной проблемной области.

На абстрактном уровне элементы динамических вычислительных структур можно рассматривать как операторы. Наиболее простой такой структурой

является *последовательность операторов*, выполняющихся друг за другом (рис. 4.8, *а*).



а) *б*) *в*) *г*)

Рис. 4.8. Основные структуры управления

При выполнении оператора *ветвления* проверяется некоторое условие и в зависимости от результатов проверки управление передается на одну из двух ветвей алгоритма (см. рис. 4.8, *б*), одна из этих ветвей может быть пустой.

Для многократного выполнения некоторых частей алгоритма используются операторы цикла. При выполнении *цикла с предусловием* вначале проверяется условие выполнения цикла, и если ответ положительный, то управление передается на выполнение последовательности операторов, составляющих *тело цикла* (см. рис. 4.8, *в*). Цикл с предусловием может не выполняться ни одного раза.

При выполнении *цикла с постусловием* вначале выполняются операторы, составляющие тело цикла, после чего проверяется условие цикла (см. рис. 4.8, *г*). Если условие цикла удовлетворяется, то тело цикла выполняется повторно и так до тех пор, пока условие выполнения цикла истинно. В отличие от цикла с предусловием, цикл с постусловием выполняется хотя бы один раз.

Другими абстрактными динамическим структурами являются переключатель, блок и подпрограмма. *Переключатель* представляет собой оператор, после выполнения которого управление передается на одну из *n* ветвей. *Блок* представляет собой абстрактную динамическую структуру любой сложности, имеющую только один вход и только один выход. *Подпрограмма* представляет собой блок, который может многократно вызываться из разных мест программы и после выполнения которого управление передается на следующий после *точки вызова* оператор (рис. 4.9). Использование подпрограмм позволяет экономить оперативную память и уменьшать затраты на программирование.

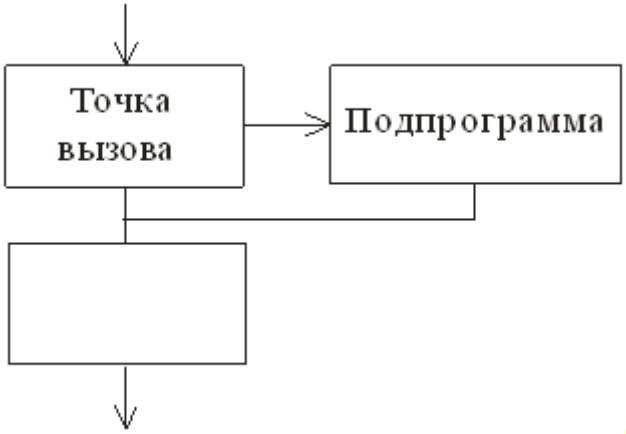
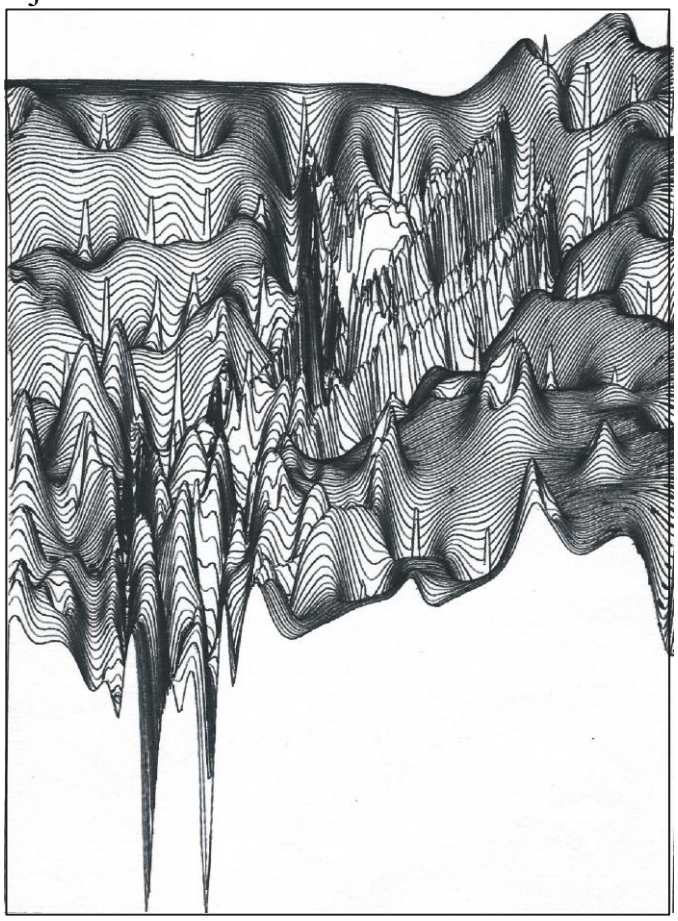


Рис. 4.9. Вызов подпрограммы

В теории алгоритмов доказано, что алгоритм любой сложности может быть представлен с помощью последовательности операторов, ветвления и циклов.

При разработке программного обеспечения программист должен учитывать своеобразие машинной арифметики, ее отличия от обычной арифметики, используемой человеком. Так, если имеются три целочисленных переменных i = 3, j = 7 и k = 4, то в результате вычисления выражения i × j / k = 5, где / – операция целочисленного деления. Но при изменении порядка выполнения операций умножения и деления можно получить ответы: i / k × j = 0 и j / k × i = 3.



Подобные случаи могут происходить в реальной жизни. Так, при разработке автором подсистемы моделирования топографических поверхностей программа первоначально была написана на языке ФОРТРАН с использованием целочисленной арифметики, а затем была переписана на языке ассемблера другим программистом, не знавшим алгоритма. По неизвестной причине этот программист переставил местами операции целочисленного умножения и деления. В результате выполнения этой программы была получена поверхность, представленная на рис. 4.10.

Рис. 4.10. Последствия ошибки в программе

4.7. Система информационного моделирования геопространства

Систему информационного моделирования географического пространства как подмножества геосистем можно рассматривать как частный случай системы с конечным числом состояний. Тогда системой геоинформационного моделирования Σ будем называть упорядоченный набор $\Sigma = (A, Q, Z, \alpha, \beta)$, где A – множество допустимых входов, в том числе первичных моделей; Q – пространство состояний системы; Z – множество допустимых выходов, включающее и вторичные модели; α – функция перевода системы в новое состояние $\alpha : A \times Q \to Q$; β – функция выхода $\beta : A \times Q \to Z$.

Такое определение системы геоинформационного моделирования дает возможность ее различной интерпретации, введения нескольких уровней рассмотрения. Границы системы однозначно определяются рамками изучаемой проблемы.

На самом нижнем уровне, соответствующем минимальному составу, система Σ_1 ограничивается процессами обработки информации на ЭВМ, начиная с ввода исходных данных и заканчивая выводом результатов в форме, пригодной для их восприятия человеком либо техническими средствами переработки информации. Этот уровень представляет интерес преимущественно для разработчиков системы. Для пользователей его знание необходимо в той мере, в какой это требуется для квалифицированной и эффективной эксплуатации программного обеспечения и всей системы Σ в целом.

На втором уровне система Σ_2 рассматривается как совокупность технического, программного, информационного, методического и организационного обеспечения, ориентированного на получение геоинформационных моделей, а также карт и планов. В такой трактовке система Σ является предметом изучения разработчиками и пользователями преимущественно в организационном плане.

На третьем уровне система моделирования Σ_3 включает коллектив разработчиков системы, косвенным образом участвующих в процессе моделирования. На этом уровне интерес представляют отношения между разработчиком и пользователем, такие вопросы, как внедрение разработанной системы, доводка системы в процессе эксплуатации (включая сопровождение программного обеспечения), модификация системы в связи с изменениями в топографо-геодезическом и картографическом производстве и т. п.

При создании автоматизированных систем обработки информации наибольшая доля затрат приходится на разработку программного обеспечения. В процессе функционирования таких систем их эффективность определяется преимущественно качеством программного обеспечения. Поэтому далее система геоинформационного моделирования будет рассматриваться в основном на первом уровне.

4.8. Системы

Особенность современных взглядов на методологию разработки и реализации проектов в самых различных областях человеческой деятельности состоит в системной ориентации или системном подходе. Под *системным подходом* понимают совокупность наиболее общих методологических принципов и методов исследования и разработки сложноорганизованных объектов, независящих от природы последних. В более общей формулировке системный подход трактуется как *современная форма* научного и технического мышления.

До середины XX столетия системный подход развивался преимущественно в рамках философии. Его истоки можно обнаружить в сформулированном Аристотелем тезисе о том, что целое больше суммы частей. Позднее это свойство систем получило название *синергетического эффекта.* Противоположностью системного подхода считается механицизм XVII–XIX вв. Сущность механистических концепций, высказанных Галилеем и обоснованных Ньютоном, заключается в том, что целое определяется свойствами его элементарных частей.

В основе системного подхода лежит требование рассмотрения частей в их взаимосвязи с целым. Исследованием существующих систем занимается *системный анализ*, а проектированием систем – *системотехника*, понимаемая как теория проектирования сложных систем. Теоретической и методологической основой системного подхода служит общая теория систем, называемая также *системологией*.

Наиболее законченной формой системного подхода служит общенаучный *принцип системности.* С системных позиций множество элементов, образующих целостный объект, рассматривается во всем многообразии их свойств, а также внутренних и внешних отношений. С системным подходом тесно соприкасается *теория систем,* или *общая теория систем,* возникновение которой обычно связывают с именем П. фон Берталанфи. Основное содержание общей теории систем как научной конкретизации общенаучного принципа системности составляет *абстрактный подход* к изучению систем.

Любая система представляет единство элементов определенной природы и формы их организации в целостный объект, поэтому возможны два пути исследования систем. В первом случае можно двигаться от изучения отдельных элементов, их свойств и отношений к пониманию свойств целого – структуры системы. Такой подход называют *методом* «*снизу вверх*». Второе направление («*сверху вниз*») связано с движением от изучения структуры систем в самом общем виде к изучению их отдельных элементов, с движением от абстрактных систем к конкретным. Наиболее общие свойства систем и служат предметом изучения общей теории систем.

Основными задачами общей теории систем являются:

- разработка средств представления или описания сложных объектов как систем;
- разработка обобщенных и специфических моделей систем и их свойств;

– исследование структуры теорий систем и системных концепций.

Обычно отмечается, что в общей теории систем широко распространен описательный подход, общеметодологические рассуждения с акцентом на развитие и уточнение понятийного аппарата и терминологии. Одновременно указывают также на тенденцию к изучению абстрактных систем с использованием формальных математических методов и на трудности их применения для описания объектов с очень большим числом переменных.

В [26] проводятся сравнение и противопоставление понятий множества и системы. Основой этого противопоставления является первичность элемента или первичность целого, что выражается в формулировках «многое, мыслимое как целое» и «целое, мыслимое как многое». Первое определение характеризует множество, а второе относится к понятию системы. В понятии множества постулируется существование элементов, не зависящее от их группировки во множество. Иными словами, в понятии множества подразумевается первичность элементов.

В системе наблюдается противоположная ситуация: целое «предшествует» своим компонентам. В теории систем предполагается, что существование целого дает возможность проводить его расчленение, выделение в нем компонентов. Способ такого разбиения целого на компоненты не является единственным и зависит от целей изучения системы.

К противопоставлению системы и множества можно также добавить то, что возможность объединения элементов в множество основана на предположении их *сходства*, тогда как возможность расчленения системы на составные части базируется на предполагаемом *различии* ее подсистем.

В настоящее время не существует общепринятого формального определения системы. С наиболее общей точки зрения *система* представляет собой произвольное отношение над абстрактными множествами *X* и *Y:*

$\Sigma \subset X \times Y$.

Если *X* и *Y* рассматривать как множество входов и выходов, то данное определение можно трактовать как отображение Σ множества входов на множество выходов

$\Sigma : X \to Y$.

Недостатком данного определения считается то, что в нем игнорируется фактор времени, а также то, что оно не дает представлений о внутреннем механизме преобразований входов в выходы. Поэтому приведенное определение характеризуют как *внешнее описание,* или как описание «*черного ящика*»*.*

Динамическая система определяется как восьмерка

$\Sigma = \{T, T_X, T_Y, X, Y, U, W, Q\}$,

где T – множество линейно упорядоченных элементов (время);

T_X, T_Y – подмножества *T*, время поступления соответственно входных и выходных сигналов;

X, *Y* – входной и выходной алфавиты;

U, *W* – множества допустимых входных и выходных значений;

Q – системообразующий критерий (отображение U в W).

Считается, что придавая определенные свойства элементам системы, можно получить систему с нужными характеристиками.

Еще одним формальным определением системы служит выражение

$$\Sigma S = \{X, Y, (R_i)\, i \in I, (f_j)\, j \in J\},$$

где X – множество входов;

Y – множество выходов;

$(R_i)\, i \in I$ – семейство отношений на X и Y;

$(f_j)\, j \in J$ – закон поведения системы.

Эти и аналогичные формальные определения подвергались критике, в связи с чем высказывались мнения, что лучшим выходом была бы тщательно составленная формальная аксиоматика. Пока что большинство авторов дают определения системы на содержательном уровне. Так, академик А.И. Берг под системой понимал организованное множество взаимосвязанных структурных элементов, выполняющих определенные функции. Обычно *система* понимается как множество элементов, связанных между собой определенными отношениями. Однако, наличие тех или иных отношений характерно для любого множества, и совокупность изучаемых отношений в системе не является раз и навсегда заданной, а определяется поставленными целями. Поэтому в дальнейшем следует иметь в виду, что в системе множество элементов связано между собой отношениями более высокого порядка по сравнению с их отношениями к любым другим элементам.

Из такой трактовки понятия системы следуют четыре условия, которым должно отвечать любое образование, рассматриваемое как система:

- целостность;
- существование критерия выделения системы – системообразующего параметра;
- наличие окружающей среды или какой-либо большей системы;
- возможность расчленения системы на более мелкие составные части или *подсистемы*.

Первые два требования взаимно обусловлены. Та или иная совокупность элементов может рассматриваться как целое лишь благодаря существованию критерия выделения системы. Последние два требования можно объединить и рассматривать как *свойство иерархичности систем,* в соответствии с которым любая система входит в еще большие и содержит еще меньшие системы.

С понятием системы тесно связано понятие окружения. Для конкретной системы *окружение* является множеством элементов, не принадлежащих системе, изменение свойств которых влияет на систему, и свойства которых, в свою очередь, изменяются в результате функционирования системы. При такой формулировке возникает вопрос о целесообразности разделения системы и ее окружения. Решение этого вопроса неоднозначно и определяется выбором критерия выделения системы. Это не значит, что система не существует объективно, но означает возможность изучения некоторого образования в

различных аспектах, то есть одно образование может рассматриваться как различные системы. Противопоставляя систему и окружение, необходимо иметь в виду, что система существует не только в окружении, но и благодаря окружению, и что система может рассматриваться как подсистема окружения.

К наиболее общим характеристикам систем, независящим от их природы, относятся:

- интегративное свойство;
- структура;
- целостность;
- изменчивость;
- связность;
- сложность;
- централизация;
- устойчивость;
- адаптируемость.

Интегративным свойством системы называется такое ее свойство, которым не обладает ни один элемент системы, взятый в отдельности.

Структура является обязательным атрибутом любой системы и понимается как форма связи между элементами системы и способ их организации в систему. Структура системы связана с ее другими характеристиками: целостностью, изменчивостью, централизацией, сложностью и т. д. В первую очередь системы характеризуются уровнем своей организации. Считается, что система тем больше организована, чем больше у нее возможность противодействовать внешним возмущениям относительно достижения выбранных целей.

Целостность системы – это ее внутреннее единство, ее отдифференцированность от окружения. Целостность системы определяет ее специфику, уникальность. Изменение структуры системы ведет к изменению самой системы, ее целостности. В связи с превращениями целостности системы возникает вопрос о границах, когда система все еще остается самой собой. Способность системы изменяться в некоторых пределах с сохранением своих основных качеств называют *изменчивостью*.

Централизация системы связана с присущей системам иерархической организацией, когда одна или несколько подсистем играют доминирующую роль в функционировании системы.

Сложность любой системы является проявлением структуры ее элементов и способа, которым эти элементы связаны. Различают *структурную* (*статическую*) и *динамическую* сложность. В настоящее время не существует единого мнения, что считать мерой структурной сложности. В качестве таковой предлагаются иерархическая структура, схема связности, многообразие элементов и сила взаимодействия.

С «иерархической» точки зрения *структурная сложность* характеризуется числом уровней иерархии в системе. Важное свойство иерархического подхода – возможность создавать надежные системы из ненадежных элементов.

Схема *связности* элементов в системе определяет пути передачи вещества, энергии или информации между подсистемами и, следовательно, определяет взаимное влияние подсистем друг на друга. Схема связности хорошо согласуется с интуитивными представлениями, что в сложной системе степень взаимозависимости подсистем выше, чем в менее сложных системах. В основе метода определения сложности системы по многообразию ее компонент лежит сформулированный Эшби *принцип необходимого разнообразия*, когда система рассматривается как отображение множества входов в множество выходов. Сложность системы при этом отождествляется с ее способностью к преобразованию многообразия входов в многообразие выходов.

Определение сложности как относительной *силы взаимодействия* между различными подсистемами и уровнями иерархии напоминает второе определение сложности (как схемы связности), но носит менее четкий характер.

Таким образом, система может быть сложной с одной точки зрения и довольно простой с другой. Выбор меры сложности зависит от целей самой системы и целей анализа. Как правило, структурной сложности системы соответствует и ее динамическая сложность. Но если система является динамически сложной, то в структурном отношении она может быть простой.

Различают также сложность неуправляемых систем и сложность управляемых систем. *Сложность неуправляемых систем* есть результат статической и динамической сложности. *Сложностью управляемой системы* называют сложность вычислений, необходимых для полного управления системой. В свою очередь, сложность вычислительного процесса характеризуется сложностью компонентных вычислений.

При разработке систем необходимо представлять, каким образом те или иные решения отразятся на их сложности, так как обычно сложность – это именно та проблема, с которой, прежде всего, сталкиваются разработчики систем.

Зависимость между сложностью систем и сложностью ее компонент выражается *аксиомами связности,* не зависящими от меры сложности [12]. Сложность системы Σ обозначается как $\Theta(\Sigma)$.

Аксиома 1. Подсистема не может быть более сложной, чем система, то есть

$$\Theta(\Sigma_i) \le \Theta(\Sigma),$$

где Σ_i является подсистемой Σ.

Аксиома 2. При параллельном соединении подсистем

$$\Sigma = \Sigma_1 \oplus \Sigma_2 \oplus \ldots \oplus \Sigma_n$$

сложность системы равна наибольшему значению сложности

$$\Theta(\Sigma) = \max \Theta(\Sigma_i) \;\; 1 \le i \le n.$$

Аксиома 3. При последовательном соединении подсистем

$$\Sigma = \Sigma_1 \otimes \Sigma_2 \otimes \ldots \otimes \Sigma_n$$

сложность системы определяется как

$\Theta(\Sigma) = \Theta(\Sigma_1) + \Theta(\Sigma_2) + ... + \Theta(\Sigma_n)$.

Аксиома 4. При соединении двух подсистем с обратной связью

$\Sigma_1 \leftarrow \Sigma_2$

сложность соединения выражается как

$\Theta(\Sigma_1 \otimes \Sigma_2) \le \Theta(\Sigma_1) + \Theta(\Sigma_2) + \Theta(\Sigma_1 \leftarrow \Sigma_2)$.

Связность является самым существенным свойством системы, так как при отсутствии связей образование уже не может рассматриваться как система. Описание связности системы может быть получено с помощью различных формальных подходов. Наилучшие результаты были получены при использовании теории графов и алгебраической топологии. В случае использования теории графов связность системы может быть представлена при помощи такого простого объекта, как матрица инциденций.

Устойчивость системы связана с ее динамическим поведением или движением. Под *устойчивой* понимается система, способная сохранять почти неизменным свое поведение. Теория *классической устойчивости* изучает изменения в поведении системы под воздействием внешних возмущений. *Устойчивой по Ляпунову* называют систему, траектория движения которой при возмущениях изменяется в незначительных пределах. *Ассимптотически устойчивой* называется система, возвращающаяся к точке равновесия через достаточно большой промежуток времени или при $t \to \infty$.

Предметом изучения теорий *структурной устойчивости* являются качественные изменения в поведении системы в результате структурных изменений в самой системе. При этом поведение системы сравнивается с поведением других, «близких» к ней систем. Если поведение изучаемой системы почти не отличается от поведения близких систем, то система является *структурно устойчивой*. Иначе, структурно устойчивой называют систему, малые структурные изменения которой вызывают малые изменения в траектории ее движения.

Адаптируемость системы является мерой ее жизнестойкости, или выживаемости, и связана с понятием устойчивости. Под *адаптируемостью* понимается способность системы изменять свою внутреннюю структуру с целью предупреждения разрушения системы или снижения оптимальности ее функционирования в результате внешних воздействий.

Существует множество признаков классификации систем. По интенсивности обмена со средой системы подразделяются на *открытые* и *замкнутые.* В зависимости от поведения систем их делят на *статические* и *динамические*, а последние – на *обратимые* и *необратимые.* В зависимости от происхождения различают *естественные* и *искусственные* системы.

По уровню организации системы подразделяют на *целенаправленные*, *адаптивные* и *самовоспроизводящиеся*. Данная классификация систем является самой интересной. Наиболее простыми и поэтому наиболее изученными целенаправленными системами являются регуляторы, по поводу которых Р. Конант и У.Р. Эшби заметили, что хороший регулятор системы должен быть

моделью этой системы. Л.А. Заде считал, что любая система является адаптивной, и что проблема только в том, чему и в какой степени она адаптивна.

Самоорганизующейся системой называют систему, которая стремится улучшить характеристики своей работы при достижении заданной цели и осуществляет это без помощи извне [13].

На самом верхнем уровне организации находятся самовоспроизводящиеся системы. Первоначально этот термин появился в биологии, что, вероятно, вполне закономерно, если учесть свойства живых организмов. В [13] в качестве примера самовоспроизводящейся системы приводится живая клетка – сложная система, производящая множество макромолекул. За время жизни клетки каждая макромолекула в ней возобновляется примерно 10^4 раз. «Но в течение всего процесса клетка сохраняет свои отличительные свойства, связность и относительную независимость. Она производит мириады компонент, но все же не производит ничего, *кроме самой себя*. Хотя за время жизни клетка объединяет по меньшей мере 10^9 различных составных молекул, она сохраняет свою индивидуальность и отличительные свойства. Сохранение единства и целостности в то время, как сами компоненты непрерывно или периодически распадаются и возникают, создаются и уничтожаются, производятся и потребляются, и называется самовоспроизведением» [13, с. 398]. К этому можно только добавить, чему равно число клеток в живом организме.

Не столь впечатляющим примером самовоспроизводства является любое предприятие или учреждение, если иметь в виду неизбежные увольнения и прием на работу новых сотрудников, или все человечество в целом, где также наблюдается постоянная «текучесть кадров».

4.9. Технические системы

Для нас наибольший интерес представляет класс технических систем. В широком смысле *техника* понимается как совокупность искусственно созданных материальных средств социальной деятельности человека. При определении предмета геодезии, картографии или геоинформатики обычно отмечается их связь с естественными науками, науками о Земле, то есть делается акцент на познавательных моментах. При этом остается в тени их активный, преобразовательный характер. Однако, не только прикладная геодезия, но и перечисленные выше области знания являются, в конечном счете, техническими дисциплинами. Поэтому при изучении и разработке систем геомоделирования представляется целесообразным рассматривать их, прежде всего, как технические системы или технические средства.

Техническое средство есть искусственное материальное образование, предназначенное для удовлетворения определенной общественной потребности. Под *потребностью* понимается напряженное состояние или несоответствие между обществом и средой. Наиболее существенной особенностью технического средства является его *действие,* направленное прямо или косвенно на удовлетворение некоторой потребности. Среди других свойств технического средства можно отметить конструктивность, системность,

синтетичность, принцип действия, единство абстрактного и конкретного, актуальность, технологичность, оптимальность и надежность [27].

Конструктивность технического средства указывает на его искусственное происхождение и функциональное назначение. *Системность* проявляется в том, что результатом технической разработки должно быть создание действующей системы, а не воспроизведение какой-либо одной стороны явления или физического закона. *Синтетичность* заключается в том, что в техническом средстве, как правило, реализованы результаты различных естественных и технических наук. *Принцип действия* – это способ соединения элементов и естественнонаучных законов в новую, не существующую в природе систему. В техническом знании принцип действия часто служит системообразующим признаком. *Единство абстрактного и конкретного* проявляется в необходимости применения абстрактных, идеализированных понятий и закономерностей к реально существующим объектам. *Актуальность* служит проявлением соответствия теоретических возможностей и практических потребностей общества, а *технологичность* подразумевает возможность воспроизведения в реальности абстрактных объектов. *Оптимальность* обусловлена необходимостью лучшего соответствия между техническим средством и потребностью. *Надежность* означает способность технического средства действовать заданным образом в заданных условиях.

Техническая система – это комплекс взаимосвязанных технических средств, предназначенных для преобразования вещества, энергии и/или информации. Важнейшими свойствами технической системы являются не только *связи,* но и *преобразования.* Преобразования могут быть связаны с *перемещением* и/или *переработкой* объекта преобразований. *Переработка* заключается в изменении внутренней или внешней структуры объекта преобразований с целью изменения его свойств.

В зависимости от объекта преобразований, технические системы подразделяются на *массовые* (вещественные), *энергетические* и *информационные.* Однако такое разделение систем носит относительный характер, так как в чистом виде перечисленные системы не встречаются. Реальные технические системы всегда представляют некоторую комбинацию вещественных (массовых), энергетических и информационных систем.

Системный подход и общая теория систем образуют теоретическую основу системного анализа. *Системный анализ* – это совокупность методов и средств, используемых при исследовании и разработке сложных объектов. Область системного анализа, занимающаяся изучением существующих, преимущественно искусственных, систем, носит название *исследования операций.* Область системного анализа, проблематику которой составляют методы разработки технических систем, называют *системотехникой.* Так как основным предметом нашего рассмотрения является разработка систем геомоделирования, а не исследование существующих, то далее будут рассматриваться только те аспекты системного анализа, которые связаны с областью системотехники.

Основными принципами системного исследования являются:

– четкая формулировка целей функционирования системы и строгое подчинение всех операций по анализу, оценке, управлению и конструированию систем главным целям;

– принцип необходимого разнообразия, ориентирующий на учет всех факторов, существенных для системы с точки зрения целей ее функционирования;

– выбор или разработка системы критериев для оценки качества системы и критериев оптимальности, соответствующих целям функционирования;

– принцип свободы выбора решения при заданном наборе естественных ограничений;

– использование специфических системных методов.

Одной из основных причин проникновения системного анализа в технические науки служит развитие производительных сил и изменение масштабов человеческой деятельности. Увеличение размеров создаваемых систем сопровождалось таким возрастанием их сложности, что стали говорить о возникновении *феномена сложности.* Появление такого эффективного средства, как системотехника, можно рассматривать как реакцию человека на возрастание уровня сложности создаваемых систем. Другой причиной системного подхода в технике является необходимость учета отдаленных последствий, обусловленных функционированием технических систем.

В настоящее время нет единства в понимании самых различных аспектов системотехники. Далее трактовка предмета системотехники, ее целей и задач дается в соответствии с [24], где проблемы системотехники рассматриваются наиболее широко. При этом учитывается тот факт, что любая техническая система проходит шесть этапов в процессе своего существования, называемых в совокупности *жизненным циклом*:

– научное исследование;
– проектирование;
– конструирование;
– изготовление;
– эксплуатацию;
– ликвидацию.

Технология разработки новых изделий включает следующие стадии жизненного цикла технических систем:

– научные исследования;
– системотехнику;
– проектно-конструкторские работы.

В отечественной практике процесс разработки новых изделий принято разделять на два этапа: научно-исследовательские работы и опытно-конструкторские работы. Таким образом, системотехнические работы если и выполняются, то распределяются по двум другим стадиям. Такой подход можно оценить как непонимание важности и недооценку этапа проектирования систем.

Основное содержание системотехнических работ, вне зависимости от специфики конкретных проектов и их сложности, составляют операции:

- уяснение задачи;
- выбор целей;
- синтез систем;
- анализ систем;
- выбор наилучших альтернатив;
- планирование действий.

Выбор и проектирование систем составляют основное содержание системотехнических работ, а проектирование деталей понимается как *конструирование*. Однако, область действия системотехники не исчерпывается только вторым этапом, а перекрывается во времени с остальными. Схема системотехнических работ состоит из пяти фаз.

Фаза системных исследований. Содержанием данной фазы является исследование конкретной области окружения и оценка будущих и текущих проектов в соответствии с полученными результатами исследования. Цель системных изысканий – согласование совокупности реализуемых и разрабатываемых проектов и создание информационного задела.

Фаза исследовательского планирования. В данной фазе усилия сосредотачиваются на конкретном проекте. Если потребности в разработке системы достаточно очевидны, работа над проектом может начинаться с этой фазы, минуя предыдущую.

Фаза планирования разработки. Содержанием фазы является более детальная проработка альтернатив, отобранных в предыдущей фазе. В операционном отношении эта фаза служит повторением фазы исследовательского планирования. Цель фазы – формулировка задач и способов их решения.

Фаза изысканий в ходе разработки. Работы, выполняемые на этой фазе, заключаются в корректировке проекта с учетом результатов, полученных на этапе проектно-конструкторской разработки и в процессе продолжающейся фазы системных изысканий.

Фаза текущих изысканий. Эта фаза занимает промежуток времени с момента окончания разработки системы и до окончания ее эксплуатации. В терминах обработки данных этой фазе соответствует сопровождение программного обеспечения.

Каждая из перечисленных фаз состоит из шести операций, соответствующих общему процессу решения задач и называемых *операциями системотехники.*

Постановка задачи. Любое исследование возникает в связи с неопределенной ситуацией, требующей решения. Поскольку решение зависит в известной мере от формулировки задачи, постольку необходимо четкое представление о ее сущности. Зависимость между формулировкой и решением задачи заключается в том, что «число возможных решений возрастает вместе с общностью и широтой формулировки и убывает с ростом ограничений и

запретов на нее» [24, с. 104]. Хотя любая проблемная ситуация характеризуется непределенностью, последняя не является абсолютной. Поэтому решение задачи начинается с поиска определенных элементов ситуации.

Системотехника предлагает два метода проведения исследований: исследование окружения и исследование потребностей. *Исследование потребностей* заключается в определении наибольшего множества потребностей, для удовлетворения которых целесообразна разработка системы. Все множество потребностей может быть разбито на четыре группы: расширение выполняемых функций, улучшение технических характеристик, снижение стоимости и улучшение внешних качеств. *Исследование окружения* состоит в изучении его состояния и прогнозировании его развития и сводится к наиболее широкому поиску теорий, принципов, методов, идей, схем, материалов, конструкций и т. д., которые могут быть использованы для удовлетворения потребностей. Наиболее существенным требованием к обоим методам является широта изучаемой области, так как именно она определяет результативность. Различие между методами состоит в том, что в первом случае идут от потребностей к окружению, а во втором – от окружения к потребностям. Но в некоторых случаях трудно отнести отдельные этапы к тому или иному методу ввиду тесной взаимосвязи между ними.

Постановка задачи может рассматриваться с точки зрения определения граничных условий. К граничным условиям относятся:

- факторы, имеющие существенное значение для решения проблемной ситуации;
- границы между системой и ее окружением;
- ограничения, под влиянием которых из множества всех решений выделяется подмножество осуществимых, приемлемых и допустимых решений.

Выбор целей. В обычном понимании цель означает «предвосхищение в сознании результата, на достижение которого направлены действия» [22, с. 406]. В системотехнике множество выбранных целей является абстрактной системой, которая есть не что иное, как описание выбранной реальной системы. Поэтому выбор целей имеет принципиальное значение при разработке систем. «Выбрать не ту цель – значит решить не ту задачу; выбрать не ту систему значит просто выбрать неоптимальную систему» [24, с. 115]. По мнению академика В.С. Пугачева, выбор критериев для сравнения различных систем, имеющих одинаковое назначение, как и любой другой вопрос о выборе критерия, не может быть решен средствами математики, и поэтому он должен решаться с позиций здравого смысла, с учетом назначения системы и условий ее применения. В зависимости от выбора конкретного критерия решение задачи определения оптимальной системы будет различным.

Важными свойствами целей являются их иерархичность и связь со средствами. *Иерархичность целей* означает, что одни цели представляют интерес не сами по себе, а как средство достижения более высоких целей. Связь *целей* и *средств* обуславливает необходимость совместного изучения целей и средств их достижения. Средства более тесно связаны с системой, чем цели; одна

и та же цель может достигаться различными средствами, а одно средство может способствовать достижению нескольких целей.

Выбранный критерий эффективности и оптимизации системы должен:

- соответствовать целям функционирования системы;
- быть состоятельным, то есть быть в состоянии оценить систему согласно цели ее функционирования;
- быть простым и иметь прозрачный физический смысл;
- быть устойчивым на определенном отрезке времени;
- быть единственным;
- быть количественным.

Процесс выбора целей имеет творческий характер и сопряжен с определенными затруднениями. Трудности в выборе целей объясняются двумя обстоятельствами. Первое из них связано с тем, что процесс выбора целей и процесс разработки системы в значительной степени перекрываются во времени. В процессе разработки новой системы (особенно – не имевшей аналогов в прошлом) почти неизбежно появление неучтенных или переосмысливание известных факторов, что влечет за собой пересмотр целей. Это связано с двумя причинами. Первая из них заключается в том, что техническое средство может служить источником информации, под воздействием которой изменяются наши представления о проблемной области. С другой стороны, разработанное техническое средство изменяет окружение. Представления о новых свойствах окружения базируются, как правило, на результатах экстраполирования его прежнего состояния. Естественно, что фактическое состояние в общем случае не будет совпадать с ожидаемым.

Эффектным примером технического средства как источника информации может служить история разработки гаечного ключа для работы в космосе. В процессе его создания разработчики учли, как представлялось, все требования. Но, как оказалось, за исключением одного и самого главного. При работе с гаечным ключом в земных условиях человек использует собственный вес. Но именно этот фактор и не был учтен.

Трудности в выборе целей сопряжены также с отсутствием единой теории о выборе целей, но некоторые результативные методики все же существуют, например, оптимизация системы ценностей в [24]. Хотя каждой технической системе присуща уникальная система ценностей, при выборе целей должен учитываться тот факт, что существуют требования, присущие многим техническим системам: стоимость, эффективность, совместимость с другими системами, гибкость, стойкость против морального старения и т. д.

В данном выше определении цели отмечается связь цели с действием. В системотехнике выбор целей самым непосредственным образом влияет на содержание работ по реализации конкретного проекта.

Синтез систем. Синтез систем заключается в описании всех возможных систем, реализующих заданные функции, и рассматривается как кульминация процесса разработки новой системы. Целью синтеза систем служит составление

перечня гипотетических систем, проработка которых должна быть достаточной для принятия решений.

В силу ограниченности ресурсов разработчика большая часть работы при синтезе систем заключается в сборе сведений об использовавшихся ранее системах, методах, теориях, идеях, устройствах и т. п. При разработке системы, не имевшей аналогов в прошлом, или когда известные решения не отвечают новым требованиям, единственным выходом служит генерация новых идей. В процессе синтеза систем деятельность системотехника должна отвечать двум требованиям. На этой стадии системотехник должен сознательно подавлять свои критические способности, так как преждевременная критика оказывает негативное влияние на отношение других участников проекта к выдвижению новых идей. Второй задачей системотехника является стремление к получению возможно большего числа альтернативных систем. Наиболее распространенная ошибка на данной стадии состоит в том, что процесс заканчивается с появлением первого приемлемого решения. Нетрудно заметить связь этих рекомендаций с решением проблем методом мозгового штурма.

Затруднения несколько снижаются при использовании метода функционального синтеза. При функциональном подходе перечисляются граничные условия, входы, выходы и необходимые функции или операции. От функций при этом требуется только их потенциальная осуществимость, вопрос об их конкретной реализации обычно откладывается. Некоторые функции системы обусловлены назначением и природой системы (минимальные или инвариантные функции), другие – принятыми конструктивными решениями.

Главный вопрос при синтезе системы – это выделение ее подсистем или *проблема декомпозиции системы.* Важнейшими факторами, определяющими границы окружения, системы и подсистем являются *входы, выходы* и *функции.* В большинстве случаев состав подсистем определяется составом функций. Другими критериями выделения подсистем являются минимизация входов и выходов подсистем или минимизация взаимодействий между подсистемами.

Анализ систем. Содержание этапа анализа систем состоит в выведении основных *следствий альтернативных систем* и их сравнении в свете предположений, начальных целей и ограничений. Цели анализа систем достигаются выбором подходящего *метода сравнения систем,* определением границ его применимости и его умелым использованием. Сравнение систем означает сравнение следствий разных систем по отношению к конкретной цели и сравнение некоторой системы с различными целями.

Выбор оптимальной системы. Выбор наилучшей альтернативы (или альтернатив) из числа разработанных должен базироваться на использовании отлаженного механизма принятия решений. Рекомендуемая системотехникой *схема принятия решений* включает элементы:

- систему ценностей;
- множество целей (являющихся подмножеством системы ценностей);
- множество альтернатив;
- методы определения вероятностей следствий;

– критерий решения (также являющийся элементом системы ценностей).

Однако чисто логический подход к принятию решений в силу сложной природы перечисленных элементов применим далеко не всегда. Обстановка, в которой принимаются решения, может содержать различные доли риска, определенности или неопределенности. Если известно полное множество следствий и вероятности всех следствий, то принято говорить, что решение принимается в *условиях риска*. Предельным случаем условий риска являются *условия определенности,* когда вероятности следствий равны 0 или 1. Под *условием неопределенности* понимается ситуация, когда известны следствия, но не известны их вероятности. Степень определенности (или неопределенности) ситуации обуславливает выбор методов принятия решений. В общем случае методы принятия решений могут основываться на произвольном выборе, пробах и ошибках, обращении к авторитету, интуиции, этических системах или математических критериях.

4.10. Системы управления

Логическим следствием постоянного усложнения создаваемых человеком систем явилось появление их нового класса – автоматизированных систем управления, то есть систем, предназначенных для управления функционированием других систем.

Управлением иногда называют деятельность по осуществлению руководства функционированием других объектов, направленную на достижение определенных целей. В этом толковании плохо то, что управление определяется через понятие руководства, и сразу возникает вопрос, что такое руководство. Маловероятно, что человек, знающий, что такое руководство, не знает, что такое управление. Кроме того, понятие управления является иным, чем понятие руководства. Можно управлять автомобилем или функционированием другого технического устройства, но при этом едва ли можно говорить о «руководстве» ими.

Интересное и лаконичное определение управления, характеризующее его суть, было дано в [19], где *управление* трактуется как навязывание обусловленного поведения. Таким образом, если наши действия вызывают нужное нам поведение объекта, то мы им управляем. Если это условие не выполняется, то у нас нет оснований говорить об управлении.

Системой управления называют часть реального мира, не совпадающую со всем миром и включающую чувствительную, управляющую, активную, изолирующую компоненты и каналы связи (рис. 4.11). Часть мира, отличную от управляющей системы, называют *средой*. Иногда выделяют *ближнюю среду* – часть среды, непосредственно взаимодействующую с системой управления, и *дальнюю среду*, не оказывающую практического влияния на работу системы управления.

Объект управления рассматривается как пассивный элемент, хотя в действительности он может быть активным, например, это может быть управляющая система нижнего уровня. В частном случае это может быть активный объект, цели которого являются антагонистическими по отношению к

управляющей системе. Объект управления, как правило, подвергается воздействию внешней среды, и целью системы управления может являться нейтрализация таких воздействий и поддержание объекта управления в требуемом состоянии либо перевод его в такое состояние. Объектом управления может быть и ближайшая среда.

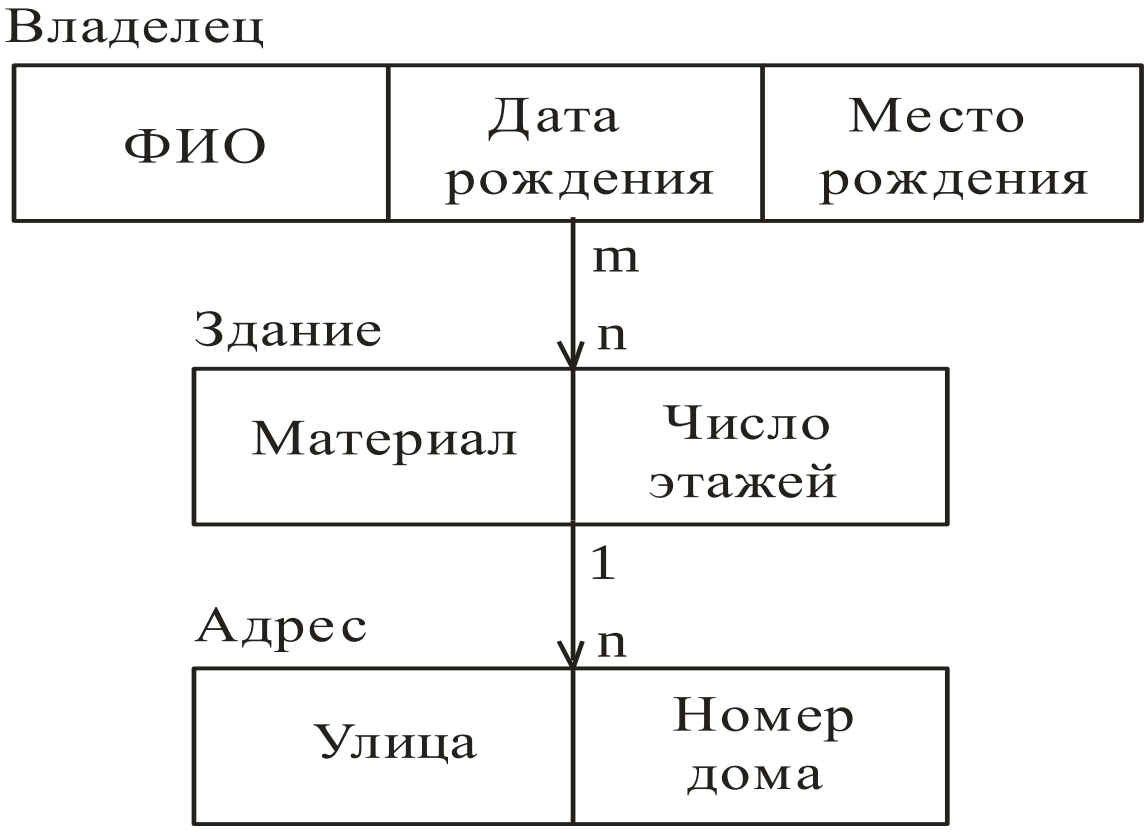


Рис. 4.11. Управляющая система

Чувствительная компонента состоит из элементов, называемых *датчиками информации*. Под воздействием объекта управления и внешней среды датчики информации изменяют свое состояние и вызывают изменения (*сигналы*) в канале связи, соединяющем их с управляющей компонентой. Таким образом, датчики фиксируют в определенные моменты времени как некоторое множество параметров объекта управления, так и, возможно, параметры внешней среды.

Управляющая компонента состоит из *преобразователя информации* и *запоминающей подкомпоненты*. Управляющая компонента принимает сигналы, поступающие от чувствительной компоненты, и преобразует их в новые сигналы, которые по другому каналу связи поступают в активную компоненту.

Активная компонента представляет собой совокупность особых элементов – *эффекторов*, функцией которых является непосредственное воздействие на объект управления.

Изолирующая компонента предназначена для защиты системы управления от воздействия внешней среды.

Тому или иному состоянию объекта управления (и внешней среды) *S* соответствует определенная совокупность сигналов *J*. Действие датчиков

информации может пониматься как отображение $F : S \to J$. Любой датчик информации характеризуется определенной чувствительностью. Поэтому любые достаточно близкие к S_0 состояния S_i (i = 1, 2, ..., n) воспринимаются системой управления как S_0, то есть имеет место $F : S_i \to J_0$.

Управляющая компонента вырабатывает последовательность управляющих сигналов, а также изменяет свое внутреннее состояние – состояние запоминающей подкомпоненты. Без запоминающей подкомпоненты поведение системы управления было бы статичным – определенному сигналу каждый раз соответствовала бы одна и та же реакция системы. Обучение и прогнозирование в такой системе были бы невозможны. Последовательность вырабатываемых сигналов формируется под влиянием принимаемых сигналов и состояния запоминающей подкомпоненты. То есть, система управления ведет себя так, как если бы внешняя среда находилась в состоянии S^*. Обозначив обратное для F отображение как Φ, можно записать, что $\Phi : J_0 \to S^*$.

Системы управления не сводятся только к техническим системам. Они широко представлены в живой природе и человеческом обществе. Управляющей системой является любой живой организм с относительно сложной организацией. Самой сложной из таких систем является человек. Среди технических управляющих систем встречаются как очень простые, так и очень сложные, насчитывающие более 10^6 элементов и получившие название *больших систем.*

Управляющие системы могут быть вложенными, иметь иерархическую организацию. Так, например, свою собственную систему управления может иметь чувствительная компонента. Функцией такой системы управления может быть управление передачей сигналов в зависимости от степени их важности, приоритета, а также поддержание работоспособности чувствительной компоненты в целом. Отдельный человек может быть объектом управления со стороны вышестоящих сотрудников организации и являться системой управления по отношению к нижестоящим.

Кроме того, одни и те же объекты могут входить в различные системы управления и играть в них разные роли. Например, конкретный человек может являться сотрудником предприятия (одна система управления), членом общественной организации (другая система управления), членом семьи (третья система управления) и т. д.

Системы геомоделирования, как правило, являются составной частью распределенных систем управления, в которых наиболее ответственные решения обычно принимаются человеком.

Представленная выше схема системы управления идеализирована, описывает абстрактную систему управления. В реальных системах управления некоторые компоненты могут иметь разную сложность своей организации (от самой примитивной до очень сложной) либо могут быть плохо выражены и даже отсутствовать вообще. В реальности существуют системы управления с компонентами, функции которых совмещены. Обязательными компонентами управляющей системы являются преобразователь информации и каналы связи.

Наличие каналов связи дает возможность передачи информации от одного компонента другому. Их отсутствие превратило бы управляющую систему в набор несвязанных элементов. *Циркуляция информации* в системе управления является признаком, отличающим систему управления от систем других типов.

4.11. Фазы обращения информации

В процессе использования информации можно выделить следующие относительно самостоятельные этапы или фазы обращения информации: восприятие, передачу, представление, хранение, обработку и воздействие.

Восприятие информации состоит в том, что в результате взаимодействия чувствительной компоненты с управляемым объектом и внешней средой изменяется состояние датчиков информации. У человека при этом происходит формирование образов объекта и/или среды. Формирование образов наблюдаемых объектов у человека является результатом его психической деятельности. Технические системы не обладают психикой, поэтому отражение объекта и среды в технических устройствах, какими являются датчики информации или запоминающая подкомпонента, можно назвать их образом только условно.

Возможность формирования образов или отсутствие такой возможности с точки зрения системы управления не имеет принципиального значения. Для управляющей системы важно то, что чувствительная компонента способна отражать среду и управляемый объект, различать его состояния.

Восприятие информации, как правило, сопровождается помехами. Поэтому в функции чувствительной компоненты может входить не только восприятие полезного сигнала, но и подавление шумов. Кроме того, измерение параметров среды или объекта управления характеризуется некоторой ошибкой измерений, определяемой чувствительностью датчиков.

Датчики информации могут предназначаться для измерения самых различных физических величин и параметров управляемого объекта. Но каналы связи предназначены для передачи сигналов вполне определенной физической природы. Поэтому перед чувствительной системой может возникнуть также задача преобразования информации к виду, пригодному для передачи по каналам связи.

Передачей информации называют ее транспортировку в пространстве. Для передачи информации могут использоваться самые различные по своей физической природе каналы связи: оптические, акустические, электрические, электромагнитные, гидравлические и т. п. В процессе передачи информации также может происходить ее искажение шумами. Поэтому выбор каналов связи осуществляется как с учетом их помехозащищенности, так и с учетом их *пропускной способности* – количества (объема) информации, передаваемого в единицу времени. Прием информации может трактоваться как ее вторичное восприятие. Передачу информации от объекта управления к системе управления называют *обратной связью.*

Представление информации является совокупностью правил кодирования сообщений с помощью сигналов. Задача представления информации

обусловлена необходимостью тем или иным способом отражать состояние объекта управления. Особенностью информации, как некоторой сущности, является то, что она не может быть выделена в «чистом виде», и проявляется только в материально-энергетической форме сигналов. Представление информации является не только отдельным этапом в ее жизненном цикле, но и способом ее существования. В процессе передачи или обработки информация всегда должна быть *представлена* тем или иным способом. Представлением информации называют также процесс ее преобразования к другому виду.

Главным критерием при выборе способа представления информации служит эффективность предполагаемых операций передачи или обработки. Поэтому преобразование (частный случай обработки) информации из одного представления в другое является весьма распространенной операцией над информацией. Суммарные затраты на представление информации в другом виде и на ее последующую обработку (передачу) часто оказываются ниже, чем только на ее обработку (передачу) в первоначальном виде.

Обработкой информации в общем смысле слова называют любое преобразование, выполняемое с целью изменения ее формы или содержания либо с целью получения ответа на тот или иной поставленный вопрос. В узком смысле обработкой информации иногда называют только решение поисковых или вычислительных задач.

Обработка информации играет ключевую роль в функционировании систем управления. Если управляющие системы без передачи информации невозможны, то без обработки информации их существование лишено всякого смысла.

Хранение информации обусловлено необходимостью ее транспортировки во времени. Хранение информации позволяет создавать «историю» управляемого объекта и самой системы управления, осуществлять анализ тенденций, строить прогнозы и т. п. Хранение информации осуществляется с помощью запоминающих устройств, имеющих различную физическую природу. В качестве запоминающих устройств можно рассматривать записную книжку (обычную и электронную), справочники, картотеки, магнитные носители информации и т. п. Запоминающие устройства характеризуются своей *емкостью*, *временем поиска информации*, *стоимостью хранения единицы информации*, *надежностью хранения*. Важной характеристикой запоминающих устройств является *время хранения информации*. По времени хранения запоминающие устройства подразделяются на *оперативные* (когда информация хранится лишь на этапе решения задачи), *постоянные* и *долговременные*.

Воздействие состоит в выработке сигналов, вызывающих определенные регулирующие, управляющие и защитные действия, целью которых являются необходимые изменения либо предотвращение нежелательных изменений в объекте управления.

4.12. Данные и информация

Понятие информации является основным понятием в управлении и системах управления. Его важность обусловлена тем обстоятельством, что не

только производственная, но и любая целенаправленная деятельность человека в принципе невозможна без управления и использования информации.

Термины «данные» и «информация» являются соотносимыми понятиями и часто рассматриваются как синонимы. Самые краткие определения данных и информации даются примерно в таком виде: «Данные – разрозненные факты» и «Информация – организованные и обработанные данные». Но эти определения, как будет видно из дальнейшего, могут быть оспорены.

В основательном учебнике, выдержавшем шесть изданий, известный специалист по системам баз данных К. Дейт не дает определения данных, ограничившись замечанием: «Термины ″данные″ и ″информация″ трактуются в этой книге как синонимы. Некоторые авторы предпочитают отличать эти два понятия, используя термин ″данные″ для ссылки на значения, которые сохранены в базе данных, а термин ″информация″ для пояснения смысла этих значений пользователю. Разница, безусловно, существенная, но предпочтительно сделать ее более определенной там, где это уместно, вместо того чтобы полагаться на различные понятия между двумя, по существу, одинаковыми терминами» [9, с. 16]. Пояснение или разъяснение – это либо *интерпретация*, установление смысла и значения символов, либо *экспликация*, уточнение смысла терминов естественного или формального языка.

В толковом словаре по информатике данные определяются как «информация, представленная в виде, пригодном для обработки автоматическими средствами при возможном участии человека» [20, с. 76]. Далее информация трактуется как «содержание, присваиваемое данным посредством соглашений, распространяющихся на эти данные; данные, подлежащие вводу в ЭВМ, хранимые в ее памяти, обрабатываемые на ЭВМ и выдаваемые пользователям» [20, с. 129]. Здесь стоит обратить внимание на неоднозначность определения. С одной стороны, информация – это некоторые данные (см. вторую часть определения), а с другой – их содержание. Но очевидно, что данные и их содержание (смысл) суть разные понятия. Кроме того, приведенные определения содержат явную рекурсию: данные – это информация, представленная в определенном виде, а информация – это данные, обрабатываемые на ЭВМ.

В философском словаре информация определена как «некоторые сведения, совокупность каких-либо данных, знаний», что совпадает с обыденным пониманием информации, когда термины «данные» и «информация» не различаются [22, с. 134].

В фундаментальном математическом словаре данные определяются как «факты или идеи, выраженные средствами формальной системы, обеспечивающей возможность их хранения, обработки или передачи. Такую формальную систему называют языком представления данных; синтаксис этого языка – способом представления информации; его семантику или прагматику – информацией» [18, с. 816]. Это определение содержит глубокий смысл и за ним стоит целая система современных взглядов, идей и концепций. В том же словаре информация определена как *содержание* сигнала или сообщения, сведения, рассматриваемые в процессе их передачи или восприятия. Впервые

такое истолкование информации было предложено «отцом кибернетики», американским математиком Н. Винером (1894–1964), различавшим *сигнал* и *информацию* – содержание сигнала.

Различие между сигналом и информацией в том, что сигнал является материальным носителем информации. Информация не может существовать отдельно от сигнала и обладает определенной инвариантностью по отношению к сигналу: одно и то же содержание может быть представлено различными сигналами.

Напомним, что любая формальная система является системой знаков. В логике знаком, или символом, называют эмпирический объект, используемый с целью указания на другие объекты: эмпирические (индивидные) – чувственно воспринимаемые и обладающие пространственно-временными характеристиками или абстрактные – целостные мысленные образования, выступающие в качестве непосредственного содержания человеческого мышления.

Таким образом, можно заметить определенную аналогию между парами понятий: используемыми в логике терминами «знак» и «значение», с одной стороны, и употребляемыми в теории связи терминами «сигнал» и «информация», с другой стороны.

Далее данные будут трактоваться как наборы символов, а информация – как содержание символов.

4.13. Измерение информации

Понятие информации является одним из основных в кибернетике, изучающей общие свойства систем управления в технических устройствах, живых организмах и социальных организациях. Представления об информации обычно ассоциируются с некоторыми сообщениями, и такое ее понимание, отождествление со сведениями любого характера существовало до появления средств связи и вычислительной техники. Развитие техники связи потребовало использования точных количественных способов оценки объемов передаваемых сообщений.

В статистической теории информации предполагается, что получателю информации известен полный *перечень альтернатив* (*алфавит*) и их *вероятности*. В 1928 г. Р. Хартли предложил численную меру передаваемой информации:

$$I(N) = \log_2 N,$$

где I – количество информации;

N – число равновероятных событий (сигналов).

Данное выражение следует понимать в том смысле, что в сообщении из N равновероятных сигналов содержится количество информации, равное I.

В 1948 г К. Шенноном была разработана математическая теория связи, в которой он уточнил понятие количества информации и обобщил формулу Хартли как

$$I = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i,$$

где I – количество информации;

p_i – вероятность i-го сигнала;

k – количество возможных сигналов.

Оценка количества информации по Шеннону имеет вероятностную основу, что вполне объяснимо. Понятие вероятности может быть использовано всюду, где имеется некоторая *неопределенность*, как это имеет место при *получении* сообщений. Передача информации неразрывно связана со снятием неопределенности, в противном случае такая передача была бы лишена всякого смысла. Снятие неопределенности всегда сводится к выбору одной или нескольких альтернатив из совокупности всех возможных вариантов, каждый из которых характеризуется некоторой вероятностью.

Работы К. Шеннона оказали столь значительное влияние на теорию связи, что ее нередко называют *шенноновской*. Но вскоре было установлено, что оценка количества информации по Шеннону применима далеко не во всех коммуникативных ситуациях. В соответствии со статистическим пониманием информации ее получатель должен иметь набор некоторых рабочих гипотез. Однако получение новых сведений в процессе коммуникации вовсе не требует обязательного выдвижения их получателем каких-либо предположений о сообщениях.

Получатель информации может принять сообщение, о существовании которого он не выдвигал никаких гипотез и даже не мог знать. В сатирической повести М. Ларни «Четвертый позвонок» некоему школьному учителю вода из колодца показалась странной на вкус, и он отправил ее на анализ, после чего получил сообщение из проводившей анализ лаборатории: «Ваша лошадь больна диабетом».

В настоящее время наиболее общими трактовками понятия информации являются две. Первая из них – истолкование информации с использованием понятия неопределенности – была описана выше. Вторая трактовка информации была предложена У.Р. Эшби в середине 1950-х гг. и связана с понятием *разнообразия.* По Эшби, передача информации возможна тогда, когда есть разнообразие. И по каналу связи нельзя передать информации больше, чем количество имеющегося разнообразия.

В соответствии с неопределенностной трактовкой информация передается лишь там, где есть неопределенность. *Неопределенность* понимается как возможность некоторого числа альтернативных исходов, каждому из которых соответствует та или иная вероятность. Неопределенность устраняется или уменьшается в результате получения определенных сообщений или сведений – информации. Таким образом, в основе статистического или вероятностного подхода к объяснению природы информации лежит взаимосвязь между неопределенностью и вероятностью.

Использование неопределенностной концепции информации за пределами теории связи сопряжено, однако, с определенными затруднениями. Очевидно,

что часто получателю информации не известен перечень альтернатив (как человеку, слушающему телевизионную сводку новостей), иначе каким образом получались бы новые знания. Объяснение этого факта было дано в предложенной Эшби *концепции разнообразия.* По Эшби, информация присутствует только там, где существует различие (неоднородность, разнообразие) хотя бы двух объектов в некотором выделенном отношении. С позиций концепции разнообразия теория информации понимается как дисциплина, изучающая процессы передачи разнообразия.

Если сравнивать обе концепции, то можно сказать, что неопределенностная концепция информации рассматривает процесс передачи разнообразия со стороны приемника (получателя), а противоположная ей концепция разнообразия – со стороны источника (отправителя, передатчика). Таким образом, в идее разнообразия подчеркивается объективный характер информации. Такая трактовка понятия информации в большей степени соответствует обыденным представлениям об информации как сведениям или сообщениям об объектах, явлениях, их свойствах и т. п.

С позиций *теории отражения* при взаимодействии объектов свойства одного объекта накладывают отпечаток на другой объект, или «воспроизводятся» в нем. Такое воспроизведение понимается как передача информации. Таким образом, с позиций теории отражения информация понимается как *отраженное разнообразие,* а передача информации – как *отражение разнообразия.*

Концепцию разнообразия (объяснение информации как разнообразия в структуре объекта) можно считать частным случаем трактовки информации как отраженного разнообразия. Понимаемую так информацию можно разделить на потенциальную и актуальную. Под *потенциальной информацией* (или *структурной*) понимается разнообразие структуры реального объекта, под *актуальной* – разнообразие отражения его структуры. Очевидно, что актуальная информация всегда меньше потенциальной, и любое описание реального объекта всегда сопровождается его огрублением.

Трудности с использованием понятия количества информации при решении проблем, отличных от задачи передачи сообщений, явились стимулом для создания других трактовок и теорий информации. В частности, была дана термодинамическая трактовка понятия информации как отрицательной энтропии – меры неупорядоченности систем. А.Н. Колмогоровым была предложена идея алгоритмического подхода к оценке информации, в соответствии с которой количество информации определяется минимальной длиной программы для однозначного преобразования одного объекта в другой. (Трактовка не очевидная, поскольку длина программы остается одной и той же, а объекты могут быть различной сложности.)

Недостатком *неопределенностной трактовки* информации является ее субъективный характер, поскольку разные получатели информации могут строить различные гипотезы в отношении вероятности получения каждого сообщения. Однако количество информации, передаваемой по каналу связи, не зависит от ожиданий ее получателя.

Разнообразностное понимание информации не требует привлечения каких-либо гипотез для оценки количества передаваемой информации. Поэтому объективный характер разнообразностной трактовки информации является, безусловно, ее сильной стороной. Другое ее достоинство – согласие с интуитивным пониманием информации как некоторых сведений или сообщений.

Статистическая (или неопределенностная) трактовка информации называется также *синтаксической*, поскольку ограничивается изучением структурно-синтаксических аспектов сообщений, рассмотрением только отношений, существующих между отдельными символами сообщений. И нужно сказать, что неопределенностное понимание информации при передаче сообщений по каналам связи вполне уместно, поскольку при этом передаются последовательности символов, а не смысл сообщений.

Таким образом, синтаксическая оценка количества информации характеризует сообщения исключительно с точки зрения их структуры как последовательностей знаков. Вне пределов ее изучения остаются качественные свойства сообщений. Рассмотрим простой пример. Пусть требуется передать по каналу связи число 12. Его можно передать как десятичное число 12, как двоичное число 1 100, как число XII в римской системе счисления, как слово «двенадцать» или слово «дюжина». Очевидно, что со статистической точки зрения в каждом из приведенных вариантов количество информации будет различным, но с точки зрения получателя сообщений их содержание будет идентичным.

Данный пример можно продолжить. Пусть требуется передавать сведения о датах некоторых событий. Мы можем представлять месяцы, как минимум, тремя различными способами: с помощью названий (январь, …, декабрь), с помощью арабских (1, …, 12) или римских чисел (I, … , XII). Более того, мы можем пронумеровать дни года и в сообщениях указывать год и номер дня в году. Статистическая оценка количества информации будет давать различные значения для каждого случая, хотя принципиальная разница в *содержании* сообщений будет отсутствовать.

Постепенно усложняя примеры, мы придем к необходимости передачи произвольных текстов на естественном языке и необходимости оценки количества содержащейся в них информации. Но на любом этническом (естественном) языке каждую мысль можно выразить, как правило, с помощью различных слов. И статистическая оценка количества информации в тексте – функция от длины текста – не всегда соответствует его реальному содержанию. Кроме того, одни и те же сообщения на различных этнических языках характеризуются разной длиной. В частности, английские тексты, как правило, короче русских.

Было многократно показано, что при коммуникации количественный аспект информации менее важен, чем содержание сообщений. Для воспроизведения сообщений необходимо каким-то образом определять количество их содержания. Изучение содержания сообщений, определение «*количества смысла*» является предметом *семантических теорий* информации.

Семантические концепции информации описывают смысловые характеристики сообщений.

Принципиальное отличие *семантической оценки количества информации* от синтаксической в том, что содержание синтаксической информации определяется структурой ее материальных носителей, исключительно синтаксическими отношениями, тогда как содержание семантической информации зависит от структуры возможных *состояний предметной области* и семантическими свойствами материального носителя информации.

Один из подходов к измерению количества информации с качественных позиций состоит в следующем. Каждый человек обладает некоторой совокупностью знаний, компетентностью в некоторой предметной области. Эти знания тем или иным образом организованы, имеют определенную структуру, называемую *тезаурусом*. Было предложено считать, что информация в сообщении присутствует только тогда, когда сообщение приводит к перестройке тезауруса, переосмыслению предметной области получателем сообщения. Количество информации при этом определяется объемом и глубиной изменений в тезаурусе. Можно отметить, что такая трактовка информации может быть распространена и на автоматизированные системы, основанные на обработке знаний.

Кроме того, с позиций получателя сообщений информация обладает определенной *ценностью* или *полезностью*, так как может быть тем или иным образом использована. В этом заключается *прагматический аспект* информации. В управляющих системах и при целеполагающей деятельности человека *ценность информации* оказывается ее наиболее важным свойством. Для управляющей системы или лица, принимающего решение, важно снятие только той неопределенности и отражение только того разнообразия, которые способствуют достижению определенной цели. Для измерения ценности информации были предложены разные подходы. Наиболее известна трактовка ценности информации, данная А.А. Харкевичем: *ценность информации* определяется приращением вероятности достижения цели после получения информации. При таком понимании информации становится возможным ее отрицательное количество, что соответствует уменьшению вероятности достижения цели. Примером может служить взаимная дезинформация сторон, участвующих в том или ином конфликте.

4.14. Информационные системы

Выше отмечалось, что отражение среды в сигналах системы управления (иначе – образ совокупности сигналов) называют *информацией.* Считается также, что информация существует только внутри систем управления.

Важнейшим звеном в системе управления служит управляющая компонента. *Информационные системы* (ИС), понимаемые как технические системы по переработке информации, являются компонентами систем управления. Иногда информационные системы называют также *системами обработки данных* (СОД).

Система управления определяет функции информационных систем: восприятие, обработку, хранение, передачу и представление информации. Управляющая компонента представляет совокупность информационных систем, выполняющих перечисленные функции. Назначением конкретной информационной системы могут являться лишь некоторые из указанных функций. Характерным свойством системы управления служит ее информационная замкнутость. Система (или орган) управления, связанная с объектом управления линиями прямой и обратной связи, образует замкнутый *контур управления,* по которому циркулирует информация. Информационные системы могут быть разомкнутыми, когда источник и потребитель информации являются различными объектами.

Как правило, информационные системы являются человеко-машинными системами или антропотехническими комплексами. В зависимости от распределения объемов работ, выполняемых человеком и техническими средствами, различают неавтоматизированные, механизированные, автоматизированные и автоматические системы обработки данных. В *неавтоматизированных системах* механизированной обработке подвергаются только простейшие, но трудоемкие для человека операции. В *механизированных системах* обработка осуществляется отдельными подсистемами, но связь между подсистемами поддерживается человеком. *Автоматизированные системы* – это системы обработки данных, в которых функции человека заключаются в сборе данных, анализе результатов и завершающей обработке. *Автоматическими* называют системы, в которых в обязанности человека входят только подготовка системы к пуску и контроль за ее работой.

Информационные системы классифицируются по организации входных данных, по организации информационного потока через систему и по организации процесса обработки.

По *организации входного потока* ИС делят на системы с пошаговой обработкой и системы с групповой обработкой. В *системах с пошаговой обработкой* данные поступают в систему ограниченными порциями, обрабатываются и выдаются потребителю в той же последовательности. В *системах с групповой обработкой* обрабатывается и поступает к получателю вся совокупность данных; пошаговая обработка в них не может быть применена, так как совокупность данных образует единое целое. Примером пошаговой обработки может служить обработка аэроснимков на цифровых фотограмметрических станциях, когда единицей (порцией) данных является стереопара. Примером задачи с групповой обработкой является уравнивание сплошных геодезических сетей: ее уравнивание не может выполняться, пока не будут введены и обработаны результаты всех измерений.

По *организации информационного потока через информационную систему* различают поточную и непоточную обработку. В *системах с поточной обработкой* данные состоят из сравнительно однородных элементов, образующих один поток. В *системах с непоточной обработкой* данные отличаются большим разнообразием, входной поток разделяется на ряд однородных групп, каждая из которых подвергается специфической обработке.

Примером непоточной обработки может служить моделирование ситуации и моделирование рельефа в существующих геоинформационных системах.

В зависимости от *способа организации процесса обработки* различают системы с комплексной и фрагментарной обработкой. В *системах с комплексной обработкой* совокупность необходимых преобразований выполняется за один запуск системы обработки данных. В *системах с фрагментарной обработкой* процесс обработки данных разбивается на ряд последовательных независимых подпроцессов, а полная обработка выполняется за несколько пусков системы.

Практика показывает, что во многих областях создание автоматических систем либо невозможно вследствие ограниченности сегодняшних знаний, либо нецелесообразно по экономическим соображениям. Наиболее эффективными и наиболее распространенными в настоящее время являются автоматизированные информационные системы.

Под *автоматизированной информационной системой* (АИС) понимают антропотехнический комплекс для автоматизированной переработки информации, ориентированный на конкретную область применения. Средства автоматизации переработки информации, входящие в систему, представляют собой различные виды обеспечения: техническое, математическое, программное, информационное, методическое и организационное.

Под *техническим обеспечением* понимается совокупность технических средств, служащих для получения, передачи, обработки, представления и хранения информации в процессе функционирования информационной системы.

В настоящее время отсутствует единая точка зрения на понятие «математическое обеспечение». Иногда выделяется программное обеспечение как совокупность программ и программной документации для реализации целей и задач цифровых электронных вычислительных машин. Иногда математическое обеспечение трактуют как синоним программного обеспечения и к нему относят всю совокупность программ, разработанных и используемых для конкретного типа ЭВМ.

Данное ниже определение математического обеспечения близко по содержанию аналогичным определениям, например, в [3]: под *математическим обеспечением* подразумевается совокупность математических методов, алгоритмов, программ и программной документации, предназначенных для организации подготовки и прохождения задач через вычислительную систему. Таким образом, понятие математического обеспечения шире понятия программного обеспечения. Содержанием математического обеспечения являются как возможные (потенциальные) методы, так и реализованные в виде программ (актуальные). Программное обеспечение включает в себя лишь последние.

Традиционно математическое обеспечение подразделяется на общесистемное и специальное или прикладное. *Общесистемным математическим обеспечением* называется совокупность алгоритмов и программ, обеспечивающих функционирование вычислительной системы независимо от содержания процессов обработки. *Специальным*

математическим обеспечением называют совокупность формальных методов, алгоритмов и программ, предназначенных для содержательной обработки информации с помощью вычислительных систем.

В функции общесистемного математического обеспечения входит:

− связь между человеком и техническими и программными средствами информационной системы;

− организация и управление функционированием средств системы обработки данных;

− изменение состава программного обеспечения в процессе функционирования системы;

− обеспечение автоматизированной разработки программного обеспечения.

Первые три функции выполняются *операционной системой,* последняя – *системой автоматизации программирования.* Выполнение любых прикладных программ на вычислительной установке производится под управлением операционной системы, так как такая организация процесса переработки информации в данное время наиболее эффективна. Иными словами, прикладные программы выполняются в среде «вычислительная система + операционная система». Изменения в любой из этих компонент неизбежно сказываются на эффективности информационной системы. Таким образом, общесистемное математическое обеспечение хотя и не определяет правил содержательной обработки информации, однако оказывает влияние на технико-экономические показатели конкретной системы обработки данных.

Информационное обеспечение есть множество данных, необходимых для решения задач информационной системы. *Организационное обеспечение* – это совокупность участвующего в технологическом процессе персонала, конкретных форм его организации и комплекса организационных мероприятий, направленных на эффективное использование всех ресурсов АИС. *Методическим обеспечением* называют совокупность документов, обосновывающих и регламентирующих способы достижения целей информационной системы. Иногда выделяют *лингвистическое обеспечение* – совокупность языковых средств, предназначенных для решения задачи представления данных в информационной системе.

4.15. Специальное математическое обеспечение

Значение перечисленных компонент автоматизированной информационной системы различно, но саму ее суть составляет специальное математическое обеспечение. Его роль настолько велика, что иногда, в узком смысле слова, информационной системой называют упорядоченный набор методов и процедур переработки информации. Специальное математическое обеспечение – это то, ради чего создаются ЭВМ, – совокупность правил содержательной обработки данных. Ни ЭВМ, ни общематематическое обеспечение правил содержательной переработки информации не определяют. Если раньше автоматизация переработки информации связывалась только с вычислительными машинами, то сейчас специальное математическое

обеспечение обычно упоминается. Но недооценка его значения продолжается довольно часто. На практике это приводит к тому, что разработка специального математического обеспечения в некоторых случаях начинается через какое-то время после установки оборудования.

Сложность математического обеспечения разрабатываемых систем такова, что на его реализацию требуется от нескольких месяцев до нескольких лет. Следовательно, к моменту окончания разработки специального математического обеспечения вычислительные системы могут оказаться близки к моральному износу. В некоторых публикациях указывается, что в настоящее время срок морального старения персональных ЭВМ составляет три года. Поэтому проблема разработки программных средств автоматизации заслуживает самого серьезного отношения и соответствующих технологий. Академиком Н.Н. Яненко для их обозначения был даже предложен термин «математическая технология».

Определение математического обеспечения как совокупности методов, алгоритмов и т. д. подразумевает его представление в виде некоторого описания. В настоящее время такие описания принято называть моделями. Класс однородных объектов характеризуется множеством состояний (*пространством состояний*), которые могут принимать отдельные объекты данного класса. *Моделью* называют обобщенную совокупность информационных параметров, служащих для описания пространства состояний выбранного класса объектов. Класс объектов может изучаться с различных сторон, его описание может преследовать различные цели. Поэтому возможно существование различных моделей конкретного класса объектов.

Процесс разработки математического обеспечения информационных систем разбивается на множество законченных этапов (некоторые авторы выделяют до 16 фаз). Далее создание математического обеспечения рассматривается как последовательность трех укрупненных этапов: разработка технологической (информационной), алгоритмической и программной моделей процесса обработки данных.

Под *технологической моделью* понимается параметрическое представление процесса автоматизированной переработки информации в АИС. Основными элементами, формирующими структуру информационной модели, являются блоки переработки информации, информационные массивы, точки диалога и параметрические связи. *Блоки переработки информации* на этапе разработки технологической модели характеризуются лишь функциональным назначением. *Информационные массивы* являются совокупностями связанных параметров, одновременно перемещаемых или хранимых в процессе функционирования информационной системы. *Точки диалога* аналогичны блокам переработки информации с той лишь разницей, что переработка информации осуществляется в них не автоматически, а с участием человека (принятие решений, корректировка результатов и т. п.). *Параметрические связи* указывают пути перемещения данных в системе, или иначе, информационное взаимодействие между элементами системы (подсистемами).

Технологическая модель является одновременно как содержательным описанием предметной области, так и моделью будущей системы обработки данных. Поэтому выбор технологической модели в то же время есть выбор информационной системы. Технологическая модель не должна содержать детальной проработки тех или иных элементов системы обработки данных, но должна отображать основные концепции, возможности и принципиальные решения.

Реальные информационные системы предназначаются для переработки информации не как абстрактной категории, а как совокупности сведений о конкретных классах процессов и статических или динамических объектов. В связи с этим на этапе разработки информационной модели должен быть решен вопрос о представлении моделируемых объектов либо объектов, служащих источником информации, – вопрос о составе и структуре информационных массивов. По существу, технологическая модель системы обработки данных отображает движение информационных массивов, их перемещения и преобразования.

Информационные массивы есть не что иное, как модели реальных объектов или совокупностей таких объектов. Разработка представления реальных объектов в АИС начинается с разработки их содержательных или «физических» моделей. *Содержательная модель* – это описание класса объектов или процессов в терминах прикладной дисциплины, изучающей выделенный класс объектов в том или ином аспекте. Содержательные модели могут быть физическими, экономическими, географическими, геодезическими, картографическими и т. п.

Следующий шаг – это *формализация содержательной модели* или *разработка математической модели* предметной области. Следует, однако, заметить, что исходные содержательные модели могут быть в той или иной степени уже формализованы в зависимости от уровня математизации соответствующей проблемной области.

Под *математической моделью* объекта понимается система математических соотношений, служащих для описания его свойств. В связи с формулировкой математической модели необходимо решение следующих задач:

– выделение элементов конкретной области реального мира (процессов, объектов, их свойств и отношений) и их структур на содержательном уровне;

– выделение абстрактных математических пространств, способных служить отображением предметной области;

– определение отношений эквивалентности между элементами выделенной области реального мира (содержательной моделью) и элементами абстрактного математического пространства;

– разработка структур элементов математического пространства, эквивалентных структурам реальных элементов.

Формализация содержательной модели начинается с формализации используемых прикладной дисциплиной понятий. С этой целью определяется система основных исходных или первичных понятий, а все остальные понятия определяются через первичные и ранее определенные. Формализация

представления объектов в автоматизированной системе обработки данных завершается созданием *информационной модели* – отображением математической модели объекта на языке технических средств автоматизации.

Таким образом, в технологической модели блоки переработки информации представляются как «черные ящики», то есть указывается лишь их функциональное назначение, но не раскрывается внутренний механизм переработки информации. Состав и содержание блоков переработки информации определяются алгоритмическими моделями. *Алгоритмической моделью* блока называют алгоритмическое представление процесса преобразования входных для данного блока информационных массивов в его выходные информационные массивы. Совокупность алгоритмических моделей всех блоков переработки информации образует *алгоритмическую модель* информационной системы. Алгоритмическая модель системы – это ее основное содержание. Качество и ценность системы обработки данных определяется качеством совокупности реализованных в ней алгоритмов – правил содержательной переработки информации.

Множество алгоритмов переработки информации передается не только от человека к ЭВМ, но и от поколения к поколению. Так или иначе, алгоритмическая модель системы отражает уровень знаний, достигнутый специалистами данной проблемной области. На качество алгоритмической модели влияют также характеристики технических средств переработки информации, затраты человеческого труда, квалификация коллектива разработчиков. Основное содержание алгоритмической модели информационной системы определяется совокупностью методов решения поставленных задач. Поскольку содержательная переработка информации является предметом изучения определенной прикладной области, постольку наряду с математиками к разработке алгоритмической модели должны привлекаться соответствующие специалисты.

Состав алгоритмической модели определяется перечнем и функциями блоков переработки информации технологической модели. Эти функции являются основными. Однако выполнение основных функций невозможно без выполнения целого ряда вспомогательных. К вспомогательным функциям АИС относятся:

- проверка корректности (правильности и полноты) исходных данных;
- реакция на ошибочные данные;
- реализация диалога с пользователем;
- обучение пользователя и оказание ему оперативной помощи;
- согласование элементов специального математического обеспечения с ближайшим окружением (техническими средствами, общесистемным математическим обеспечением, другими элементами специального математического обеспечения, требованиями пользователя);
- протоколирование процесса переработки информации, вывод и комментирование результатов.

Важнейшее требование к алгоритмической модели, вытекающее из ее сущности, – однозначность. Так как суждение о выполнении этого условия выносится человеком, то возникает второе столь же важное требование – обозримость. Так как на сегодняшний день отсутствуют языки, обладающие одновременно лаконичностью, емкостью, понятностью и однозначностью, то было предложено несколько спорное решение осуществлять представление алгоритмической модели на двух уровнях. Алгоритмическая модель первого уровня должна обеспечивать обозримость, второго уровня – однозначность.

Информационные модели предметной области и алгоритмические модели процессов обработки данных образуют исчерпывающее описание процесса содержательной переработки информации. Так как переработка информации должна выполняться техническими средствами, то последние должны понимать это описание. Отображение информационной и алгоритмической моделей информационной системы на языке технических средств переработки информации называется *программной моделью* системы. Программная модель состоит из основных и вспомогательных программных модулей. *Основные программные модули* предназначены для выполнения функций, определяемых информационной и алгоритмической моделями системы. *Вспомогательные программные модули* служат для взаимного согласования всех модулей и для сопряжения программной модели с ближайшим окружением: техническими средствами автоматизации, общематематическим программным обеспечением, другими элементами специального математического обеспечения. Назначение *основных информационных модулей* – хранение информационных массивов, необходимых для работы основных программных модулей; *вспомогательные информационные модули* содержат информацию для вспомогательных программных модулей.

Функции основных программных модулей определяются назначением информационной системы. Функции вспомогательных программных модулей заключаются:

- в управлении работой программных средств;
- в согласовании программных и технических средств;
- в контроле за работой технических средств, блокировании и ликвидации последствий аварий.

Разработка программной модели АИС заключается в представлении информационной и алгоритмической моделей на языке формализованного представления алгоритмов. Перевод с такого языка на язык технических средств (трансляция) осуществляется автоматически.

Рассмотрим решение задач с использованием ЭВМ. На рис. 4.12 представлены основные процессы и их результаты при автоматизированном решении задач, и связи между ними.

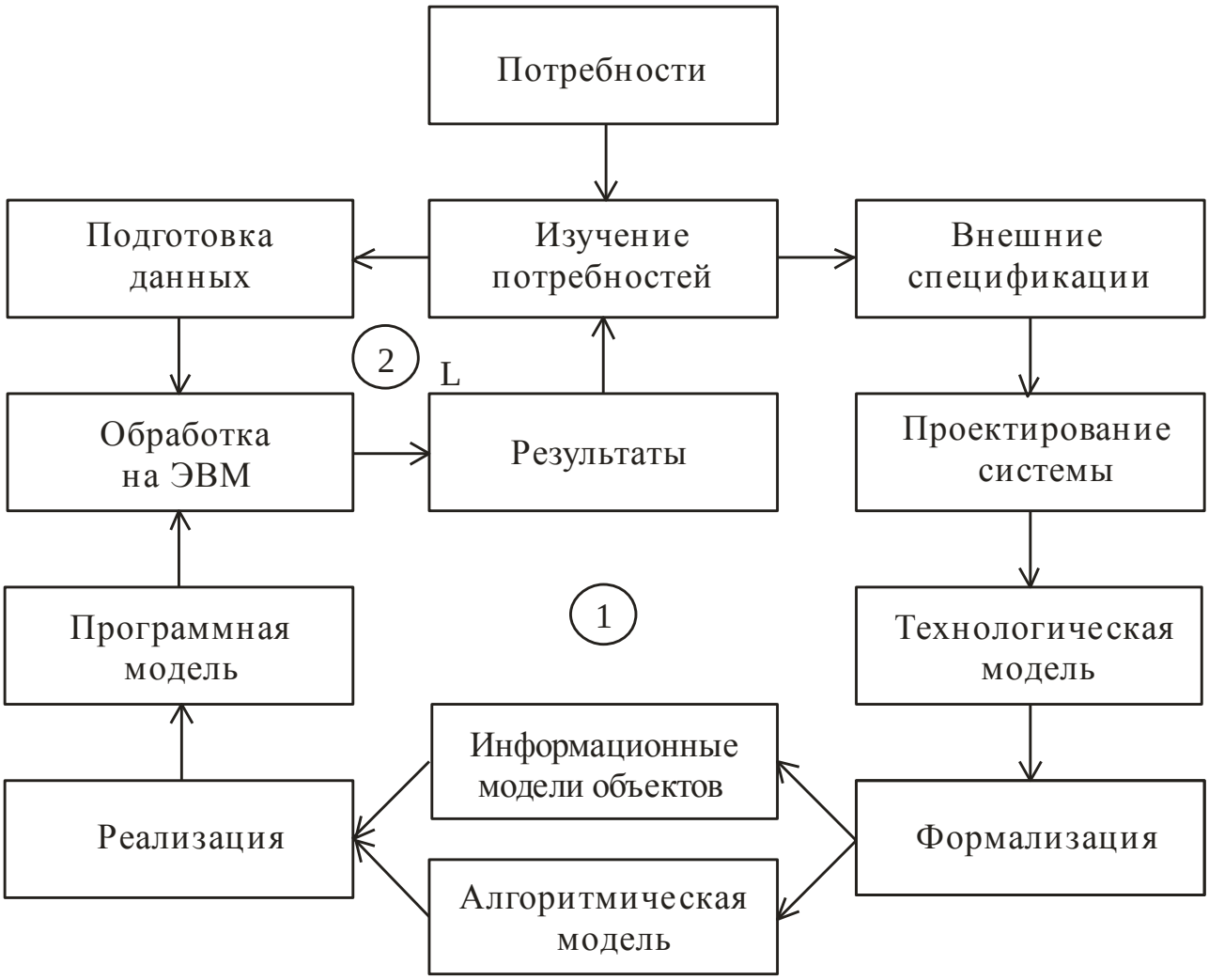


Рис. 4.12. Автоматизированное решение задач

Необходимость разработки автоматизированной информационной системы, в том числе системы геомоделирования, может возникнуть в результате заключения договора на ее разработку либо в связи с решением руководства организации о создании подобной системы для дальнейшей реализации.

В первом случае изучение потребностей сводится к интенсивному общению с заказчиком, описанию и документированию его требований к будущей системе. От пользователя требуется осмысление и формулирование своих долговременных информационных потребностей.

Этап следует считать концептуальным, так как именно на нем определяются функциональные возможности будущей программы, такие ее важнейшие свойства, как универсальность и адаптируемость. Пользователь, например, может потребовать либо работы программы с заранее предопределенными объектами моделирования, либо возможности создания и изменения нужных объектов по мере необходимости. Очевидно, что структура и размеры области применения таких двух программ, а также стоимость их создания и время жизни будут различны.

Внешние спецификации содержат основные требования и ограничения на разрабатываемое специальное математическое обеспечение. Они включают

описание содержания и форм входных и выходных документов, описание хода решения задачи, взаимосвязей между данными, методов контроля данных и реакции программы на обнаруженные ошибки, частоту или периодичность решения задачи, точность представления и максимальные объемы данных, требования к быстродействию и т. п.

Принципиальный и неизбежный недостаток подготовленного пользователем содержательного описания задачи – это то, что оно излагается на этническом языке с использованием терминов проблемной области. Естественный язык характеризуется неоднозначностью, а специальная терминология может быть неправильно понята лицом, осуществляющим формализацию задачи.

Данный этап требует тщательного подхода, так как известно, что чем раньше совершены ошибки, тем выше затраты времени и средств на их устранение и программное обеспечение в целом. В США известны прецеденты, когда вопреки согласованным многотомным спецификациям заказчики утверждали, что информационная система им не соответствует, и обе стороны тратили миллионы долларов на судебные разбирательства.

Если информационная система разрабатывается в инициативном порядке, то ее пользователем будет являться не конкретное лицо или организация, а некоторая категория конечных пользователей, выделенная из всего многообразия пользователей по типу решаемых задач. В этом случае необходимо установить такие требования к разрабатываемой системе, которые большинство потенциальных пользователей признает разумными.

Целью проектирования является получение технологической модели будущей автоматизированной информационной системы. Эта работа должна выполняться системотехниками (системными аналитиками) совместно со специалистами соответствующей проблемной области. Их задача – на основе содержательного описания проблемы разработать общее описание системы, включающее организацию данных, а также функциональную декомпозицию процесса обработки данных.

На основании представленной технологической модели математиками осуществляется построение математических моделей объектов предметной области и их отношений, а также разработка алгоритмических моделей процессов обработки информации.

Результат работы математиков – математические модели объектов и алгоритмические модели – поступает к программистам, обязанностью которых является описание процесса решения задачи с использованием формальной знаковой системы – языка программирования. Этап, названный реализацией, включает такие процессы, как собственно написание программы, ее отладка и тестирование. Результатом усилий программиста является программа в исходных и машинных кодах (исполняемый модуль) и программная документация.

Последовательность этапов изучения (уточнения) потребностей, проектирования системы, разработки и реализации математических моделей образуют замкнутый контур или цикл, который мы будем называть *циклом*

разработки / модификации программного обеспечения (контур 1 на рис. 4.12). Информация, циркулирующая в этом контуре, характеризует долгосрочные потребности пользователя.

Представленная технологическая схема разработки информационных систем сильно идеализирована. Процесс разработки программного обеспечения является итеративным из-за своей сложности; даже программу размером в одну страницу практически нельзя написать с первой попытки без ошибок. Для разработки более или менее сложных программных комплексов требуется многократное выполнение цикла 1, так как в процессе формализации и реализации могут быть обнаружены ошибки, допущенные на предыдущих этапах. Хотя при внутренней приемке (альфа-тестировании) функции заказчика выполняет сам разработчик, это не меняет итеративной природы процесса.

Основные проблемы, стоимость и сроки разработки программного обеспечения связаны с преобразованием содержательного описания решаемой задачи в код, пригодный для исполнения автоматом. Считается, что эти проблемы обусловлены принципиальными различиями между описанием, понятным для человека, и описанием, понятным для машины.

Высокая стоимость и продолжительные сроки разработки программного обеспечения являются постоянными стимулами для совершенствования методологии и технологии программирования. С этой целью были предложены методы структурного и объектно-ориентированного программирования; применяется визуальное программирование; разрабатываются новые языки программирования и совершенствуются существующие; используются отладчики и оптимизирующие трансляторы, CASE-технологии; разрабатываются библиотеки программ и классов для решения конкретных задач и т. д. Последним новшеством в сфере технологий программирования является разработка унифицированного языка моделирования UML. Несмотря на значительный прогресс в данной области, стоимость программного обеспечения продолжает оставаться высокой и часто намного превышает уровень затрат на технические средства.

Подготовка данных и их обработка на ЭВМ образуют второй замкнутый контур (цикл 2 на рис. 4.12), который мы будем называть *циклом эксплуатации* программного обеспечения. Обращающаяся в этом контуре информация имеет оперативный характер и связана с текущими потребностями пользователя. Естественно предполагать, что в общем случае изменение долговременных потребностей пользователя отражается и на его текущих потребностях.

Опыт использования программных комплексов различного назначения свидетельствует, что после перехода к циклу 2 – эксплуатации – могут требоваться неоднократные возвраты к выполнению цикла 1 с целью устранения обнаруженных ошибок либо модификации программного обеспечения. Необходимость модификации обусловлена динамичностью требований к программному обеспечению: могут изменяться как внешние условия, например, требования нормативных документов, так и представления пользователя в результате накопления опыта автоматизированного решения задач. Требования к программному обеспечению изменяются не только во

времени – у конкретного пользователя, но и в пространстве – от пользователя к пользователю. Основой непрерывного процесса развития и совершенствования автоматизированных информационных систем являются накопление опыта их использования, переосмысление методов обработки данных, появление более эффективных структур данных и алгоритмов.

Стоимость и сроки модификации программного обеспечения определяются его структурой и глубиной вносимых изменений. Принято считать, что изменение одного оператора в процессе модификации по трудоемкости эквивалентно написанию десяти операторов на стадии разработки программы. Время модификации может составлять от нескольких часов до нескольких месяцев. Необходимость модификации программного обеспечения автоматически ведет не только к увеличению его стоимости; из-за снижения коэффициента готовности математического обеспечения пользователь может нести убытки или потерять доверие заказчиков.

Хотя каждая АИС характеризуется (как техническая система) уникальностью своих качеств, можно выделить совокупность свойств, желательных для всех информационных систем. К ним относятся:

- производительность;
- надежность;
- уровень автоматизации;
- экономичность.

Важнейшими показателями являются экономичность и производительность. *Производительность* информационной системы определяется тремя факторами: пропускной способностью, коэффициентом готовности и временем ответа. *Пропускная способность* понимается как объем работы, выполняемый АИС за единицу времени, и зависит от быстродействия технических средств переработки информации и качества программного обеспечения. *Коэффициент готовности* характеризует степень готовности системы к выполнению своих функций и может быть выражен отношением

$$k = \frac{t_p}{t_o + t_p},$$

где t_o – время, необходимое для ликвидации неисправностей в технических средствах и для модификации программного обеспечения с целью устранения обнаруженных ошибок;

t_p – время нормальной работы системы.

Временем ответа называют промежуток времени между моментом получения данных для обработки и моментом выдачи результатов.

Надежность и уровень автоматизации по отношению к экономичности и производительности имеют подчиненное значение.

В многочисленных публикациях подчеркивается возрастание роли математического обеспечения и его стоимости. В середине 1960-х гг. затраты на создание технических и математических средств распределялись примерно поровну, а в конце 1970-х гг. расходы на разработку математического

обеспечения достигали 80 % от общей стоимости системы обработки данных, то есть превышали стоимость оборудования в четыре раза. Тенденция к возрастанию затрат на математическое обеспечение продолжается и ныне. На этом основании некоторые специалисты предполагают, что стоимость математического обеспечения будет превышать стоимость аппаратных средств в 10 раз.

С другой стороны, в последние десятилетия ни в одной области человеческой деятельности, вероятно, не было такого числа провалов, как в разработке математического обеспечения. «Мир программирования усеян останками программ, которые когда-то считались готовым продуктом, но впоследствии оказались неправильными, неэффективными, непонятными или непригодными по какой-то другой причине. Возможно, простейшее объяснение этого факта заключается в том, что ЭВМ – относительно новый и сложный инструмент, и требуется время для того, чтобы научиться хорошо им пользоваться. Не так легко овладеть ЭВМ как мощным инструментом для решения задач, особенно математических» [7, с. 13].

В свое время во многие учебники по программированию входила история о том, как США вынуждены были взорвать ракету, отклонившуюся от заданного курса. Как потом оказалось – из-за ошибки в программе. Эта программа была написана на языке ФОРТРАН, в котором используется так называемое правило умолчания, когда тип всех неописанных переменных устанавливается по их первому символу. В злополучной программе одна из переменных IO и I0 была ошибочной, но сработало правило умолчания, и транслятор не обнаружил ошибки. Возможно, что это была самая дорогостоящая ошибка в мире программирования. После этой истории правило умолчания подверглось всеобщему порицанию, и с тех пор в языках высокого уровня появилось обязательное условие описания всех используемых программой переменных.

Во времена «больших» ЭВМ среднее время жизни прикладных программ часто составляло немногим более года, и в некоторых случаях математическое обеспечение приобретали менее 10 % потенциальных пользователей.

Математическое обеспечение условно можно рассматривать как специфическое техническое средство, поскольку оно функционирует определенным образом и с определенной целью и характеризуется социальными эффектами. В описанной выше ситуации социальный эффект лишь один – непроизводительные затраты материальных и человеческих ресурсов. Ввиду высокой стоимости создаваемого продукта разработка АИС должна базироваться на тщательном изучении качеств (системы ценностей), которыми должно обладать ее математическое обеспечение.

Ранжирование требований к математическому обеспечению абстрактной информационной системы в порядке убывания их значимости бессмысленно, так как в реальных системах они могут следовать в какой угодно очередности. Поэтому ниже рассматриваются более или менее распространенные требования:

- разумная универсальность;
- адаптируемость;

- эффективность;
- модифицируемость;
- технологичность;
- надежность.

Универсальность математического обеспечения, понимаемая в самом широком смысле, должна рассматриваться в аспектах: среда, данные, задачи, результаты. Универсальность понимается как возможность использования математического обеспечения в различных ситуациях. Проблема универсальности возникает в связи с многообразием элементов реального мира. Поэтому универсальность системы может трактоваться как степень адекватности системы реальному миру, точнее, его выделенной части. Универсальность определенным образом связана с размерами систем и их числом или распространенностью, поскольку замечено, что в ограниченном объеме реального мира число однотипных систем (распространенность) убывает по мере возрастания их размеров [23]. Другими словами, в заданном пространстве тем меньше однотипных систем, чем больше их размеры. Поэтому стремление разработчиков придать универсальность математическому обеспечению означает стремление покрыть этим математическим обеспечением возможно более широкую область потребностей.

Однако, универсальность математического обеспечения должна ограничиваться сверху, что диктуется двумя обстоятельствами. Первое из них – это ограниченность ресурсов разработчика. Второе связано с тем, что функционирование любого универсального средства менее эффективно, чем специализированного, так как отдифференцированность, специализация функций в человеческом обществе являются основой прогрессивного развития.

В *аспекте* «среда» универсальность математического обеспечения означает *совместимость* с техническими средствами автоматизации, общесистемным и математическим обеспечением и с другими автоматизированными системами. Совместимость программного комплекса с техническими средствами – это возможность его использования на различных вычислительных системах. Необходимым условием такого использования является достаточность вычислительных ресурсов (оперативной и внешней памяти, быстродействия, внешних устройств). Отсюда следует, что в программном комплексе по необходимости должен использоваться минимум ресурсов вычислительной системы.

Так как специальное математическое обеспечение функционирует в среде «вычислительная система + операционная система», то очевидна необходимость его совместимости с общесистемным математическим обеспечением, что обеспечивается соблюдением требований и ограничений, накладываемых операционной системой.

Если информационная система взаимодействует с другими системами, то необходима совместимость их математического обеспечения. Между системами одного уровня иерархии осуществляется только информационное взаимодействие, поэтому для их совместимости достаточно *совместимости информационных баз* (иногда это одна база). Если взаимодействуют выше- и

нижестоящая системы, то кроме совместимости информационных баз требуется *совместимость по управлению.*

В *аспекте* «*данные*» универсальность математического обеспечения означает возможность обработки входных данных с различной структурой, объемом, диапазоном и точностью. В *аспекте* «*задачи*» универсальность понимается как способность специального математического обеспечения решать широкий спектр проблем в конкретной прикладной области. В *аспекте* «результаты» универсальность – это возможность представления выходных данных в различном виде по желанию пользователя.

Специальное математическое обеспечение, как и любое техническое средство, подвержено влиянию *закона ограничения многообразия*. Поэтому реализация достаточно большого числа различных алгоритмов для разнообразных видов оборудования экономически нецелесообразна. Следовательно, требование универсальности должно ограничиваться определенными рамками и дополняться свойством адаптируемости математического обеспечения. Если универсальность предполагает функционирование программного комплекса во множестве стандартных (фиксированных, предусмотренных) ситуаций, то требование адаптируемости связано с вероятностью существования непредусмотренных условий. Необходимость адаптируемости диктуется изменением потребностей и условий функционирования как в пространстве (от пользователя к пользователю), так и во времени (у конкретного пользователя).

Адаптируемость означает в первую очередь возможность формирования подмножества специального математического обеспечения (*генерацию системы*) по желанию пользователя и его модификацию в процессе эксплуатации в связи с изменениями во внешней для математического обеспечения среде (изменения в конфигурации вычислительного комплекса, в операционной системе, в потребностях пользователя и т. п.).

Особым случаем адаптируемости является *мобильность* математического обеспечения, то есть возможность переноса на другой тип ЭВМ. Хотя решение указанной проблемы характеризуется высокой сложностью, и не существует теории переноса программ, тем не менее известны определенные практические приемы, повышающие мобильность программных средств (использование языков высокого уровня и т. п.). Иногда целесообразность переноса оспаривается вследствие увеличения требуемых ресурсов (чаще всего – времени обработки) – неизбежной расплаты за универсальность любого рода. В пользу мобильности математического обеспечения свидетельствуют неуклонно увеличивающиеся затраты на программирование, высокие темпы развития вычислительной техники, а также тенденция к интеграции ЭВМ в многомашинные комплексы и вычислительные сети.

С адаптируемостью ассоциируется возможность *развития* специального математического обеспечения. В широком смысле развитие математического обеспечения может рассматриваться как любая модификация с целью оптимизации функционирования программного комплекса. В узком смысле развитием математического обеспечения называют расширение состава

выполняемых функций. Таким образом, адаптируемость означает повышение универсальности по отношению к среде, а развитие – по отношению к решаемым задачам, увеличение «мощности» математического обеспечения. Необходимость предусматривать возможность развития обусловлена ограниченностью средств на разработку отдельной системы, развитием производственной сферы (возникновением ранее не существовавших проблем), развитием знаний о средствах и методах переработки информации.

Очевидно, что способность к развитию находит свое отражение в уровне затрат на модификацию математического обеспечения, связанную с расширением функциональных возможностей, за исключением затрат на реализацию самих функций.

Наиболее важный показатель, в конечном итоге определяющий жизнеспособность математического обеспечения, – это его *эффективность,* которая понимается как затраты на переработку фиксированного объема информации. Стоимость переработки информации складывается из стоимости ресурсов ЭВМ, задействованных в процессе функционирования специального математического обеспечения. Самым дорогим ресурсом вычислительной системы часто оказывается центральный процессор. Поэтому эффективность математического обеспечения достигается эффективностью алгоритмов, использующих процессор наиболее интенсивно. Чтобы подчеркнуть указанное обстоятельство, иногда говорят о *вычислительной эффективности* алгоритма, подразумевая под этим количество операций процессора, необходимых для преобразования исходных данных в результаты. Однако, в действительности дело обстоит сложнее, и зависимость между временем процессора и эффективностью математического обеспечения не столь прямолинейна.

На общую эффективность программного комплекса оказывают влияние необходимая оперативная память, внешняя память и вычислительная эффективность. Число машинных операций можно сократить, если все многократно используемые параметры вычислять только один раз и хранить их в оперативной памяти. Оперативная память имеет хоть и большой, но ограниченный объем. В связи с использованием режима мультипрограммирования – «одновременного» выполнения нескольких программ – вся оперативная память редко выделяется одной программе. Монополизирование оперативной памяти одной программой приводит, с одной стороны, к ожиданию в очереди других программ (любая программа должна выполняться в оперативной памяти), а с другой стороны – к простаиванию других ресурсов ЭВМ, так как маловероятно, что одна программа использует все вычислительные ресурсы одновременно. Так что проблема оптимизации соотношения «быстродействие – оперативная память» существует и на современных компьютерах.

При большом числе параметров используется внешняя память. Необходимое условие такого использования: время обмена данными между программой и внешними устройствами должно быть меньше или хотя бы сопоставимо со временем их повторного вычисления. В противном случае общее время обработки возрастает и может стать критическим, несмотря на то,

что время использования центрального процессора уменьшается. Создавая программное обеспечение для конкретной вычислительной системы, разработчики в состоянии оценить затраты времени на повторные вычисления и обмен данными и принять оптимальное решение. При переносе математического обеспечения на другую ЭВМ или при изменениях на той же вычислительной установке (смене процессора или внешних устройств) может измениться соотношение между временем вычислений и временем пересылки данных. Поэтому в ряде случаев может потребоваться пересмотр ранее принятых решений.

На вычислительную эффективность программного комплекса оказывают влияние следующие факторы: выбор языка программирования, качество транслятора, тип ЭВМ (система команд процессора), операционная система, квалификация программистов, цели разработки (система ценностей), архитектура вычислительной системы, структура данных, исходные данные и организация обработки данных.

Формализация представления алгоритмов предполагает использование того или иного языка программирования – знаковой системы, предназначенной для описания процессов решения задач на ЭВМ. *Языки программирования* принято разделять на *формальные алгоритмические* и *неалгоритмические языки* и *не вполне формализованные знаковые системы* [15]. Наибольшее распространение получили формальные алгоритмические языки. Они подразделяются на следующие:

- алгоритмические языки машин и операционных систем;
- машинно-ориентированные алгоритмические языки;
- проблемно-ориентированные алгоритмические языки;
- универсальные алгоритмические языки.

Программирование на *машинном языке*, характерное для ЭВМ первых поколений, означает составление программы непосредственно в машинных кодах (системе команд ЭВМ). Крайне низкая производительность труда программистов, очень высокий процент ошибок и трудная воспринимаемость получаемых программ являются причиной того, что машинные языки уже давно не используются. *Языком операционной системы* называют алгоритмический язык, воспринимаемый комплексом «вычислительная система + операционная система». Эти языки также не применяются. Однако алгоритм, описанный на любом другом языке, в конечном итоге представляется на машинном языке либо языке операционной системы. Указанные языки обладают тем свойством, что позволяют достичь максимальной эффективности, какая только может быть достигнута на данном вычислительном комплексе. Поэтому естественно, что эффективность алгоритма зависит от системы команд и операционной системы. Другими словами, эффективность алгоритма зависит от того, насколько данная вычислительная система и операционная система подходят для решения конкретных задач обработки данных. ЭВМ вовсе не являются однородными по своему назначению. Можно выделить более или менее специализированные ЭВМ (для научных и инженерных задач, для экономических задач, для управления технологическими процессами) и

универсальные. Так, первые ПЭВМ в стандартной конфигурации часто не поддерживали операции над вещественными числами, для их аппаратного выполнения требовалась установка арифметического сопроцессора.

Машинно-ориентированные языки по своим функциональным возможностям близки к системе команд вычислительной системы, но по форме более близки к алгоритмическим языкам высокого уровня. Примером машинно-ориентированных языков служат автокоды (языки символического кодирования, языки ассемблера) и универсальные языки типа вышедшего из употребления АЛМО. Особенность машинно-ориентированных языков, прежде всего – языков ассемблера, состоит в том, что программы, написанные с их использованием, по эффективности не уступают программам на машинном языке или языке операционной системы, но обладают намного большей выразительностью. Поэтому языки ассемблера являются необходимым средством для представления алгоритмов, к эффективности которых предъявляются самые высокие требования.

Проблемно-ориентированные алгоритмические языки предназначены для решения задач обработки данных в конкретной узкой области. Примерами могут служить языки для обработки списков (ЛИСП), языки для программирования экономических задач (КОБОЛ) и научно-технических задач (ФОРТРАН, ПАСКАЛЬ), языки для разработки систем искусственного интеллекта (ПРОЛОГ) и др. Языки данного типа отличаются относительной независимостью от технических средств и удовлетворительным качеством программ.

Универсальные машинно-независимые языки характеризуются максимальной независимостью от типа вычислительной системы, наибольшей выразительностью и лаконичностью. Экспериментально было установлено, что программисты пишут за рабочий день примерно равное количество операторов, почти не зависящее от используемого языка. Поэтому использование универсальных языков позволяет достичь максимальной производительности труда программистов. Однако с повышением степени независимости от аппаратных средств отмечается снижение эффективности программ.

В разных работах приводились результаты сравнения эффективности языков программирования для научно-технических задач. Если время выполнения программ на ассемблере принять за единицу, то среднее время выполнения программ на других языках по опубликованным результатам одного из экспериментов может быть представлено табл. 4.3.

Таблица 4.3. Время выполнения программ

Язык или транслятор	Время
Оптимизирующий ФОРТРАН	1,2
Стандартный ФОРТРАН	1,8
ПЛ-1	1,5
АЛГОЛ-60	2,7

В некоторых случаях время выполнения программ на ФОРТРАНе равнялось 1, а на АЛГОЛе-60 – 16 единицам. Таким образом, в зависимости от выбранного языка программирования эффективность программного средства

может изменяться в значительных пределах, и сравнение алгоритмов (а не программ), реализованных на разных языках, довольно проблематично.

Трудности сравнительной оценки эффективности алгоритмов этим не исчерпываются. Довольно часто при сравнении различных методов обработки данных недостаточно четко разделяются понятия алгоритма и программы, реализующей данный алгоритм. Так как оценка качества алгоритма обычно производится по результатам работы программы, то возникает масса возможностей для ошибочных выводов о действительных характеристиках алгоритмов.

Для иллюстрации высказанного утверждения приведем таблицу, где в качестве примера влияния *стиля программирования* на эффективность программы рассматривалась задача вычисления элементов треугольника Паскаля [28]. С этой целью экспериментаторы составили 10 вариантов программы на ФОРТРАНе для ЭВМ IBM-360/65. При подготовке издания на русском языке тексты каждого варианта проверялись на ЭВМ БЭСМ-6 для двух различных трансляторов (Д и М).

В табл. 4.4 дано время (в секундах) выполнения вычислений по 500 раз в цикле. Полученные результаты демонстрируют условность такого понятия, как «стиль программирования», а также дают некоторые представления о качестве трансляторов. При переносе программ на другую ЭВМ или смене транслятора могут быть отсеяны эффективные алгоритмы.

Таблица 4.4. К оценке стиля программирования

ЭВМ	Номер варианта								
	2	3	4	5	6	7	8	9	10
IBM-360/365	5,8	3,9	4,0	2,2	2,1	2,0	1,4	1,2	1,2
БЭСМ-6/Д	12,1	10,0	10,1	4,2	4,0	3,4	3,6	2,3	4,3
БЭСМ-6/М	1,4	1,5	2,4	0,7	1,0	0,7	1,1	1,3	1,0

Известны результаты другого исследования влияния индивидуальных характеристик программистов на эффективность программ. Группе квалифицированных программистов была предложена алгебраическая задача. Некоторые показатели их деятельности приводятся в табл. 4.5.

Таким образом, соотношение максимального и минимального показателей составляет для времени разработки 16/1, для размера программы – 6/1, для времени вычислений по программе – 5/1. Наиболее интересным в приведенной таблице является тот факт, что практически отсутствует зависимость между общим стажем работы в качестве программиста и характеристиками программ. Однако, неоднократно отмечалось, что решающим моментом оказывается стаж работы в конкретной прикладной области. После работы в продолжение некоторого периода неизбежно появление субъективных представлений о требованиях, которым должна отвечать хорошая программа.

Таблица 4.5. Влияние индивидуальных характеристик

Программист	Стаж работы, лет	Время разработки, дней	Размер программы, ячеек	Время выполнения, с
1	2	11	4 433	3,2
2	7	76	3 215	2,0
3	11	24	1 050	3,5
4	3	58	4 957	3,5
5	5	66	4 042	7,9
6	11	111	2 460	2,4
7	4	88	2 140	3,5
8	2	19	3 550	5,0
9	10	12	1 186	3,0
10	9	60	6 137	6,6
11	8	7	2 286	2,2
12	4	24	2 690	1,6

В другом эксперименте его участники должны были сформулировать цели, преследовавшиеся ими при написании программы. Оказалось, что у каждого участника эксперимента была своя собственная система ценностей, в которую иногда даже не включалось получение программы, пригодной для работы. Противоречия в постановке целей, подходах и энергия, с которой защищаются точки зрения, нашли отражение в одной из публикаций, где утверждалось, что если бы в средние века существовали ЭВМ, то одни программисты сжигали бы других за ересь. Наконец, иногда отмечают противоречие между декларируемыми ценностями и фактическими целями.

Из сказанного можно сделать вывод, что для формирования коллектива, способного эффективно решать задачи обработки данных определенной направленности, требуется несколько лет, так как программирование в значительной степени все еще остается искусством. Вероятно, по этой причине удачными оказываются нередко лишь вторая, третья, а то и *n*-я версия программного комплекса.

В книге [21, с. 239] отмечалось, что с теоретической точки зрения «существуют задачи, не имеющие лучших алгоритмов решения» и что «для некоторых задач можно доказать невозможность получения истинных оценок сложности вычислений». Вычислительные затраты в таких задачах существенно зависят от данных; к ним, вероятно, можно отнести задачу сортировки. Если наихудшие оценки часто могут быть получены сравнительно легко теоретическим путем, то вычисление средних оценок является серьезной проблемой при высокой степени неопределенности будущих условий функционирования алгоритма и их постоянной изменчивости. Выходом из данного положения мог бы служить программный анализ совокупности данных, но такое решение не универсально, так как его проведение сопровождается дополнительными вычислительными затратами, а целесообразность может оказаться сомнительной.

Однако из этого не следует делать вывод об абсолютной невозможности сравнения вычислительной эффективности алгоритмов. Прежде всего можно попытаться теоретически установить, является ли алгоритм полиномиальным

(когда максимальное число операций возрастает не быстрее, чем полином от n, где n – размерность задачи) или экспоненциальным (в противном случае). Отбраковка неудовлетворительных алгоритмов может осуществляться, прежде всего, по этому критерию. Существуют задачи, для решения которых в настоящее время известны только неполиномиальные алгоритмы. Возможности применения таких алгоритмов чрезвычайно ограничены.

Известны также результаты экспериментов по определению жизнеспособности *метода спецификаций модулей.* В одном из таких экспериментов программный комплекс был разбит на пять модулей. Каждый модуль реализовывался несколькими способами таким образом, что разные версии одного модуля были совершенно идентичны в функциональном отношении и отличались только внутренней структурой. Путем комбинирования различных версий каждого из пяти модулей было получено 192 варианта программного комплекса. Время работы комплекса колебалось от 4.4 до 303 с. Столь колоссальное (почти семидесятикратное) расхождение в эффективности готового программного изделия свидетельствует о недостаточности описания требований к оценке качества конкретного модуля. Субъективные представления программистов о месте и роли разрабатываемых модулей в системе неизбежно ведут к тому, что полученное математическое обеспечение будет далеко от оптимального в любом смысле.

Во многих публикациях отмечалось, что обычно основное время выполнения алгоритма (около 80 %) приходится на незначительную его часть (около 20 %). Отсюда следует, что разработка отдельных модулей должна выполняться с учетом их влияния на общую эффективность.

Получение оптимального варианта математического обеспечения возможно только при тщательной проработке его архитектуры, включая структуры данных.

Под *технологичностью математического* обеспечения понимают совокупность его свойств, способствующих выполнению технологических процессов эффективным образом. Технологичность АИС является не только следствием свойств средств автоматизации (технических средств и математического обеспечения), но и эргономических требований, распределением функций между человеком и средствами автоматизации. В одной из автоматизированных систем оператор должен был перерабатывать информацию со скоростью, превышающей обычную в 19 раз. Очень высокие требования к пользователю (его квалификации, интенсивности труда, реакции и т. п.) порождают негативное отношение к самой идее автоматизации.

Технологичность математического обеспечения подразумевает простоту его использования, хорошую диагностику ошибок, гибкость, естественность, обучение пользователя. *Простота использования* предполагает минимальные усилия со стороны пользователя на освоение программного комплекса, на подготовку заданий по обработке данных и в процессе обработки, представление результатов в удобном виде. *Естественность* означает, что общение с системой обработки данных (подготовка заданий, диалог) осуществляется на языке, близком к языку пользователя. Под *гибкостью*

понимается возможность выбора методов обработки и представления результатов по желанию пользователя, возможность управления процессом обработки.

С технологичностью и эффективностью математического обеспечения связано понятие надежности. Ненадежное математическое обеспечение нельзя назвать ни технологичным, ни эффективным. *Надежность* математического обеспечения понимается как его способность обрабатывать любые наборы данных, в том числе содержащие ошибки. Источником ошибок могут служить алгоритм, программа, исходные данные и технические средства автоматизации. Поэтому надежность рассматривается в разных аспектах: алгоритмическом, программном, информационном и вычислительном [3].

Алгоритмическая надежность – способность алгоритма адекватно отображать правила содержательной обработки данных. Алгоритмическая надежность определяется тем, насколько правильно автор алгоритма мыслит процесс преобразования исходных данных в результаты и насколько верно он описывает процесс преобразования на алгоритмическом языке. Последнее обстоятельство обусловлено тем, что «существует значительный разрыв между постановкой задачи, понятной человеку, и постановкой задачи, позволяющей использовать формальные правила» [21, с. 43]. Хотя известны методы формального доказательства правильности алгоритмов, объем работ по доказательству правильности алгоритма на порядок выше затрат по его разработке. В [21] говорится о «типичном случае», когда для алгоритма длиной в 13 строк потребовалось доказательство его правильности в 150 строк. Кроме того, для доказательства правильности алгоритма требуется очень высокий уровень математической подготовки. Наконец, системы специального математического обеспечения содержат сотни тысяч и более операторов. Поэтому трудно представить не только отсутствие ошибок в доказательстве правильности, но и вообще саму возможность его завершения (по крайней мере – в разумные сроки). Вероятно, такой подход представляет интерес скорее в теоретическом плане, чем в практическом.

Альтернативой построению надежных алгоритмов методом доказательства их правильности служат метод верификации программ и исчисление программ. В основе *метода верификации* программ лежит предположение о существовании безусловно правильной документации и доказательство соответствия программ их спецификациям. Преимущество метода верификации состоит в том, что процесс доказательства формализуется и может выполняться вычислительной системой. Но в этом случае остается открытым вопрос о правильности спецификаций программы, то есть правильности представлений разработчика о процессе переработки информации. Метод верификации связан с неэффективным использованием вычислительных ресурсов. Так, верификация простой программы сортировки векторов потребовала около 0,5 часа машинного времени и сопровождалась выдачей распечатки объемом в 20 страниц. Но даже такие показатели считаются неплохими.

Исчисление программ, предложенное Э. Дейкстрой, основано на определенной дисциплине программирования, включающей в себя набор

правил для построения правильных программ. Предполагается, что *конструктивно правильная*, или *правильная по своему построению программа*, не требует доказательства своей правильности. Такая точка зрения квалифицируется как «самый замечательный принцип инженерного подхода к программированию (и резко отличающийся от современного стиля математических публикаций)» [21, с. 42].

Программная надежность является мерой адекватности отображения алгоритма на языке технических средств автоматизации. Так как разработка программного обеспечения осуществляется автоматизированным способом, то неизбежно применение трансляторов для представления алгоритмов на машинном языке. Трансляторы, сами являющиеся программами, в свою очередь могут содержать ошибки. Следствием ошибок в трансляторах могут быть ошибки в получаемых программах. Но даже абсолютно безошибочный транслятор может выдать программу, содержащую ошибки. Причиной могут служить отклонения в работе технических средств от нормального режима во время работы транслятора.

Информационная надежность математического обеспечения – это его способность обрабатывать любые наборы данных, в том числе наборы, содержащие ошибки. Исходная информация для системы обработки данных формируется человеком и/или техническими средствами. Вероятность появления ошибок может быть больше или меньше, но никогда не равна нулю. Можно, конечно, считать, что за ошибки в исходных данных ответственность несет пользователь. По статистике, человек делает одну ошибку на сто операций, и разработчик математического обеспечения не вправе требовать от пользователя безошибочной работы и должен применять «*оборонительное программирование*», опирающееся на концепции защиты и устойчивости к ошибкам. Сущность оборонительного программирования заключается в том, чтобы не только обеспечить соответствие между спецификациями программного комплекса и его использованием, но и исключить возможность его неправильного использования. Архитектура программного комплекса должна разрабатываться с учетом стратегии обеспечения защиты, и одним из обязательных компонентов программного комплекса должен быть механизм защиты. Различные аспекты оборонительного программирования были разработаны давно. Здесь отметим только, что наиболее доступным и результативным средством служит так называемый «откат» – возвращение системы к предыдущему состоянию.

Вычислительная надежность – это способность математического обеспечения правильно выполнять свои функции при различного рода отклонениях от нормального режима работы технических средств. Хотя надежность технических средств автоматизации постоянно возрастает, абсолютная надежность недостижима. «В настоящее время только невежда может считать абсолютной надежность технических средств» [10, с. 184]. Ненадежность технических средств является следствием такого свойства материального мира, как случайность.

Из всех методов проверки правильности программ наибольшее распространение получили отладка и тестирование. Цель *отладки* – получение работающего варианта программы. Задача *тестирования* – проверка результатов работы программы на совпадение с заранее известными результатами. Иногда тестирование называют доказательством *методом полного исчерпания.* Ясно, что возможности тестирования с целью доказательства правильности программ крайне ограничены. Хотя «тестирование программ может служить для доказательства наличия ошибок, но никогда не докажет их отсутствия» [8, с. 13], оно обладает одним бесспорным преимуществом. Дело в том, что все методы так или иначе ограничиваются доказательством соответствия программы ее спецификациям. Ни один формальный метод не в состоянии установить адекватность наших представлений реальной действительности. Только тестирование (правда, в редких случаях и при условии «правильности» программы) может обнаружить наличие ошибок в постановке задачи, например, когда результаты абсурдны в физическом или ином смысле.

Под *модифицируемостью* понимают возможность манипулирования текстом программы с целью получения необходимого эффекта. Необходимость внесения изменений вызвана чаще всего ошибками, которые могут быть обнаружены в процессе разработки или эксплуатации. Другой фактор – изменения в формулировке внешних спецификаций программного комплекса. «Проектирование и реализацию больших систем программного обеспечения никогда нельзя считать полностью завершенными: тот простой факт, что на создание подобных систем требуется достаточно длительное время, указывает на неизбежное отставание поставляемого продукта от ожиданий пользователей» [21, с. 248]. Третья причина – изменения среды, в которой функционирует программное обеспечение. Наконец, необходимость внесения изменений определяется опытом, появляющимся у разработчиков или у пользователя в процессе эксплуатации программного комплекса.

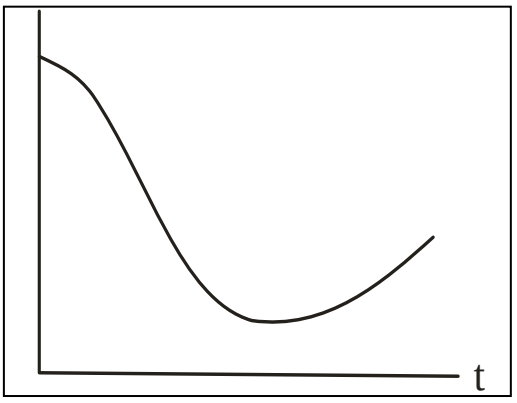


Рис. 4.13. Число ошибок как функция времени

Необходимо, однако, заметить, что развитие математического обеспечения является не столь простой задачей, как это может казаться, и не сводится к механическому расширению его функциональных возможностей. Одновременно с расширением выполняемых функций разработчикам приходится прилагать значительные усилия для поддержания программных средств в работоспособном состоянии.

В [1] приводится типичный график числа ошибок, обнаруживаемых в больших программных комплексах (САПР) с течением времени (рис. 4.13). Кривая на рис. 4.13 имеет такой же характер, как и кривая отказов технических

средств. Объяснение этого факта в следующем. В начальный период эксплуатации программного комплекса число пользователей невелико; условия эксплуатации и потребности пользователей не отличаются большим разнообразием. С ростом числа пользователей увеличивается число обнаруженных ошибок и число предложений по реализации новых функций. В результате наложения многочисленных заплат на работающие программы происходит нарушение стройности программного комплекса в целом. Эту потерю концептуального единства программного продукта иногда характеризуют как возрастание уровня энтропии в системе, в результате чего она может практически полностью утратить свою работоспособность.

Перечисленные выше требования к математическому обеспечению (универсальность, адаптируемость, эффективность, технологичность, надежностъ и модифицируемость) образуют, как это обычно бывает, противоречивую систему критериев. Требование универсальности противоречит требованию эффективности, а при определенных условиях также понятности и, следовательно, модифицируемости. Стремление к экономии оперативной памяти имеет побочный эффект – снижение быстродействия. Повышение надежности также отрицательно сказывается на эффективности, так как различного рода проверки не выполняют *полезной* обработки. Легко модифицируемая программа требует, как правило, для своего выполнения больше времени и оперативной памяти. При желании перечисление противоречий можно было бы продолжить. Но это только одна, теневая сторона проблемы взаимосвязи требований к программному обеспечению.

Другая, более светлая сторона этих отношений – возможность использовать одни свойства программного обеспечения как средство для достижения целей более высокого порядка. Адаптируемость, надежность и, тем более, модифицируемость сами по себе практического интереса не представляют. В конечном счете любое из этих качеств может явиться средством повышения эффективности. Пусть, например, для работы некоторого алгоритма требуется 1 час процессорного времени, а проверка данных для этого алгоритма занимает 10 с. Тогда осуществление проверки будет способствовать снижению общих затрат на обработку, если ошибки встречаются чаще, чем в одном из 360 наборов данных. Таким образом, затраты машинного времени на проверку данных могут быть вполне оправданными с точки зрения окончательного эффекта. Подобные вопросы должны решаться индивидуально для каждой разрабатываемой АИС. Проблема заключается обычно не в максимизации или минимизации любой ценой некоторого критерия, а в получении программного комплекса с хорошо сбалансированными характеристиками, который большинство потенциальных пользователей будет готово признать удовлетворительным. Создание таких комплексов невозможно без применения системного подхода.

Основная методологическая ошибка, допускаемая при создании больших систем обработки данных, «связана с попыткой экстраполяции опыта разработки небольших программ на проектирование программного обеспечения больших систем» [3, с. 10]. Иногда методологические ошибки имеют место

даже при разработке небольших и несложных программ. В. Турский, анализируя в [21] два случая из собственной практики, приходит к выводу о том, что в обоих случаях он не справился с решением задач, хотя составил вполне удовлетворительные и безошибочные программы.

Основной источник ошибок такого рода – ограниченность понимания поставленной задачи. Как правило, внимание концентрируется на внутренних проблемах математического обеспечения, другие аспекты АИС игнорируются. Так как «различия в целях (в учитываемых факторах) ведут к существенным различиям в программах, разработанных с учетом этих целей (факторов), нельзя рассматривать выбор как избыточное требование или предоставлять право выбора программисту» [21, с. 207]. В частности, было подмечено, что руководители предприятий или крупных проектов склонны большую часть своего времени заниматься проблемами, в решении которых они являются специалистами. Данное обстоятельство получило название *эффекта втягивания.* Предполагается, что взаимное непонимание между прикладными специалистами и новым поколением программистов, получивших чисто программистское образование, объясняется отсутствием экономической подготовки у последних. Можно также констатировать небрежное отношение программистов к специалистам, некомпетентным в области обработки данных, и нежелание вникать в их проблемы. С другой стороны, пользователи часто раздражены необходимостью объяснения «очевидных» вещей специалистам по разработке математического обеспечения.

Таким образом, созданию математического обеспечения должна предшествовать тщательная разработка основных концепций для данной информационной системы. Глубина или детальность проработки различных аспектов АИС должна быть достаточной для принятия решений о составе, структуре и функциях математического обеспечения.

4.16. Математическое обеспечение и системотехника

Обычно разработка математического обеспечения представляется следующей совокупностью этапов [11]:

- анализ требований;
- определение спецификаций;
- проектирование;
- кодирование;
- тестирование;
- эксплуатация и сопровождение.

Ниже разработка программного обеспечения будет рассматриваться с позиций системотехники и в ее терминах. Рассмотрение проблем создания специального программного обеспечения начнем со второй фазы системотехнических работ – фазы исследовательского планирования, когда необходимость в разработке АИС стала очевидной. Первым этапом работы над конкретным проектом является постановка или уяснение задачи. На данном этапе должны быть определены:

– цели пользователя, достижение которых должно быть обеспечено внедрением информационной системы;
– границы АИС;
– информационные потребности пользователя;
– требования к техническим средствам автоматизации;
– выбор технических средств;
– распределение функций между человеком и средствами автоматизации;
– факторы, влияющие на функционирование системы.

В результате выполнения этапа должен быть получен исчерпывающий перечень внешних спецификаций специального программного обеспечения.

Специфика второго этапа (*выбора целей*) заключается в том, что перечень свойств, которыми должно обладать специальное математическое обеспечение, довольно стабилен. Все перечисленные выше требования желательны для любых программ или программных комплексов. Однако их значение для каждой системы различно. Поэтому сущность данного этапа заключается в определении дополнительных специальных требований к математическому обеспечению, распределении приоритетов среди всего множества требований и формулировка конкретных ограничений.

Проектирование, рассматриваемое в литературе по программированию как единый процесс, в системотехническом плане представляет собой совокупность трех этапов: синтез систем, анализ систем и выбор оптимальной системы. *Синтез систем* начинается с разработки технологической модели АИС. Состав технологической модели был описан выше. Здесь напомним только, что она должна отражать основные концепции, принципиальные решения и возможности специального математического обеспечения, или иначе, его состав, структуру и функции.

Следующий шаг, направленный на детализацию технологической модели, – разработка содержательных моделей объектов, являющихся источниками информации, и их формализованного описания – математических описаний объектов. Для описания одного объекта может существовать несколько содержательных моделей. Это можно было видеть на примере содержательного определения понятия информации. Основным критерием в процессе выбора содержательной модели может служить ее результативность при решении задач, поставленных перед информационной системой.

В свою очередь, одной и той же содержательной модели может соответствовать несколько математических моделей. Во-первых, различными могут быть математические модели на момент «появления» объекта в системе. Во-вторых, может возникнуть необходимость представления объекта несколькими математическими моделями на разных этапах процесса переработки информации. Например, некоторая случайная величина может быть представлена выборкой из генеральной совокупности, гистограммой, законом распределения или центральными моментами первого и второго порядков и математическим ожиданием.

Ради общности, мы можем рассматривать процесс переработки информации как процесс преобразования информационных моделей. В основе такой трактовки лежит свойство транзитивности моделей. Необходимость преобразования моделей может возникать по двум причинам: либо нас интересуют сведения, содержащиеся в исходной модели в неявном виде, либо по соображениям повышения эффективности на отдельных этапах процесса переработки информации.

Поэтому параллельно с разработкой математических моделей объектов должна разрабатываться алгоритмическая модель процессов переработки информации. Алгоритмическая модель некоторого блока переработки информации может рассматриваться как описание процесса отображения одной модели объекта в другую. Такое отображение может осуществляться разными способами. Выбор метода отображения диктуется соображениями адекватности (точности, полноты и однозначности), объемом вычислений или необходимой оперативной памяти и другими характеристиками программы. В качестве примера возьмем функцию, заданную своими значениями в определенных точках. Если требуется получить ее аналитическое выражение, то, прежде всего, возможно использование аналитических выражений разного вида. Однако неединственность представления функции не исчерпывается только выбором типа выражения. Зафиксировав тип выражения и используя метод неопределенных коэффициентов, получим систему линейных уравнений. Для ее решения могут использоваться прямые или итерационные методы, в пределах которых также возможны различные варианты.

Хотя математические модели объектов являются пассивными элементами процесса переработки информации, их влияние на оптимальность системы в целом чрезвычайно велико, так как характер необходимых преобразований логически следует из состава математических моделей объектов и их свойств. Поэтому можно считать, что выбор математических моделей объектов определяет, с известными оговорками, *глобальную оптимальность* специального математического обеспечения. После того, как выбор математических моделей сделан, повышение эффективности становится возможным в сравнительно узких границах. Поэтому можно принять, что выбор алгоритмических моделей обеспечивает *локальную оптимальность.* Сказанное не относится к новым эффективным алгоритмам, не известным на момент создания системы. При появлении новых чрезвычайно эффективных методов может потребоваться пересмотр состава и структуры всей системы прикладного математического обеспечения.

Содержание *этапа анализа систем* специального математического обеспечения – оценка технологической модели, математических моделей объектов и алгоритмических моделей блоков переработки информации в соответствии с выбранной системой ценностей (критериев, целей). От математических моделей объектов зависят такие свойства системы, как потребность в оперативной и внешней памяти, время разового обмена данными между внешней и оперативной памятью, степень адекватности моделей реальным объектам.

Производными от алгоритмической модели прикладного математического обеспечения являются такие его характеристики, как общее время процессора, частота обмена данными между оперативной и внешней памятью. Как правило, для представления алгоритмической модели на языке технических средств автоматизации не требуется больших объемов оперативной памяти. Можно считать, что объем оперативной памяти, необходимой для размещения кода программы, пропорционален длине ее текста, для некоторых блоков переработки информации могут требоваться значительные дополнительные объемы памяти для хранения информационных массивов, являющихся внутренними для этих блоков. И все же утверждение о незначительных потребностях блоков переработки информации в оперативной памяти справедливо, так как выше проводилось различие между информационными массивами и блоками переработки информации.

Технологическая модель АИС определяет такие ее стороны, как универсальность, адаптируемость, надежность, технологичность, модифицируемость.

Этап выбора оптимальной системы принципиально не отличается от аналогичного этапа при разработке технических средств. Хотя выбором оптимальной системы процесс проектирования математического обеспечения формально заканчивается, фактически он может продолжаться во время разработки программной модели, а также в ходе эксплуатации и сопровождения, поскольку разработка больших систем «никогда не заканчивается».

При создании больших систем прикладного математического обеспечения полезно использование определенных принципов, установленных эмпирическим путем. Их применение не обеспечивает автоматического успеха, но позволяет повысить вероятность достижения поставленных целей по сравнению с методами бессистемных разработок.

Первым можно назвать *принцип разумной универсализации.* Следование принципу разумной универсализации подразумевает необходимость наложения ограничений на функциональные возможности системы. Введение ограничений вызвано двумя соображениями. Во-первых, это экономические соображения. Может показаться парадоксальным, но ограничение общественных потребностей, ограничение многообразия оказывается экономически выгодным. Принцип ограничения разнообразия имеет место для всех технических систем и считается достижением XX в. [10]. Во-вторых, практически невозможно предусмотреть все существующие и потенциальные ситуации и потребности. Поэтому в первую очередь реализации должны подлежать функции, соответствующие наиболее распространенным, типичным потребностям и ситуациям.

Второй принцип разработки специального математического обеспечения – *принцип подготовки развития.* Его применение вызвано теми ограничениями, которые были наложены на систему в соответствии с принципом разумной универсализации. Ограничения, вполне приемлемые в начальный период существования прикладного обеспечения, рано или поздно вступят в конфликт

либо с реальной действительностью (потребностями), либо со старыми знаниями (способами удовлетворения потребностей). Наиболее существенно на возможность развития влияет архитектура программного обеспечения.

Третий принцип – принцип *параметрической универсальности.* Следуя данному принципу, необходимо предусматривать возможность выбора пользователем значений параметров, влияющих на процесс и результаты переработки информации. Другими словами, принцип параметрической универсальности является тем средством, при помощи которого пользователь в той или иной мере может управлять процессом обработки. Ясно, что чем больше число управляющих параметров, тем шире возможности вмешательства в процесс обработки.

Однако увеличение числа управляющих параметров (и, следовательно, объемов входной информации) может приводить к возрастанию числа ошибок во входном потоке данных. Полученные результаты могут даже оказаться противоположными ожидаемым. Чтобы устранить указанное противоречие, принцип параметрической универсальности дополняется принципом умолчания. *Принцип умолчания* является соглашением, в соответствии с которым неописанным во входном потоке управляющим параметрам присваиваются некоторые предопределенные значения.

Пятый принцип – *принцип функциональной избыточности,* означающий, что одна и та же функция может выполняться различными способами в зависимости от конкретной обстановки и целей пользователя. Примером применения данного принципа может служить разработка нескольких программ сортировки или нескольких методов создания моделей топографических поверхностей.

Принцип функциональной избирательности служит дополнением к предыдущему принципу. Очевидно, что во многих случаях пользователю могут не потребоваться некоторые возможности специального математического обеспечения. Тогда их включение в систему повлечет за собой неоправданные затраты ресурсов. Следование принципу функциональной избирательности предполагает разделение функций на обязательные и факультативные и включение последних в состав математического обеспечения по желанию пользователя.

Принцип модульности признается всеми авторами, но понимается по-разному. Проблема разбиения системы на модули – это проблема декомпозиции, поэтому возможно, что общего решения не существует. Основной принцип выделения модулей – их относительная независимость. Модульность является необходимым, но не всегда достаточным условием легкой модифицируемости программного обеспечения.

Отсюда вытекает следующий принцип – *принцип наименьшего взаимодействия* между модулями, понимаемый в информационном смысле. Если целью разработки ставится высокая автономность ее частей, то о ее достижении можно судить по общему числу параметрических связей между модулями. Но интенсивность связей отражает и объективную сложность системы. Поэтому проведение принципа наименьшего взаимодействия – это

стремление к исключению связей между модулями сверх объективно необходимых.

С принципом модульности связан также *принцип иерархичности управления,* в соответствии с которым управление работой модуля осуществляется в нем самом.

Наконец, последний принцип – *принцип информационной локализованности* – состоит в том, что информация о некоторой структуре данных сосредотачивается в единственном модуле, и доступ к данным может осуществляться только из этого модуля. Результатом развития этого принципа явилась концепция объектно-ориентированного программирования.

4.17. Некоторые тенденции в программном обеспечении

Основой непрерывного процесса развития и совершенствования технических систем является накопление опыта их использования и переосмысление принципов их создания. Разработка автоматизированных картографических и геоинформационных систем в значительной степени базируется на устаревших концепциях и не учитывает современные тенденции в программировании. Чтобы пояснить суть этих претензий, рассмотрим решение задачи с использованием ЭВМ.

На рис. 4.12 отсутствуют такие сущности, как ЭВМ и ОС, так как они не предназначены для решения конкретных прикладных задач. Мы можем считать, что обработку данных выполняет прикладная программа, так как с момента запуска программы в зависимости от ее назначения универсальная ЭВМ может превратиться в «машину для начисления заработной платы» или в «машину для моделирования геопространства» и т. д.

Процесс осмысления и формулирования пользователем своих долговременных информационных потребностей требует тщательного подхода, так как известно, что чем раньше совершены ошибки, тем выше затраты времени и средств на программное обеспечение. Этап следует считать концептуальным, так как именно на нем определяются функциональные возможности будущей программы, такие ее важнейшие свойства, как универсальность и адаптируемость.

Основные проблемы, стоимость и сроки разработки программного обеспечения связаны с преобразованием содержательного описания решаемой задачи в код, пригодный для исполнения автоматом. Считается, что эти проблемы обусловлены принципиальными различиями между описанием, понятным для человека, и описанием, понятным для машины.

Рассматривавшиеся выше методы повышения производительности труда разработчиков программного обеспечения используются и в процессе его модификации, но не играют решающей роли. Более кардинальный характер имеют методы снижения затрат на сопровождение программного обеспечения, называемые *новой информационной технологией* и направленные не на повышение производительности труда системных аналитиков и программистов – посредников между пользователем и ЭВМ, а на их полное,

по возможности, исключение из процесса модификации программного обеспечения и передачу этой функции конечному пользователю.

Новая информационная технология основана на новом понимании автоматизированного решения задач: процесс решения начинается не с момента ввода данных в ЭВМ, а с возникновения потребности в решении задачи. Такой подход означает, что объектом оптимизации должно быть не только или не столько время непосредственного решения задачи на ЭВМ, сколько *коэффициент готовности* программного обеспечения *K*, определяемый как

$$K = \frac{T - T_н}{T},$$

где T – общее время нахождения программы в эксплуатации;

$T_н$ – время, в течение которого программа не может использоваться по своему назначению.

Новые информационные технологии предполагают изменение не технологии программирования, а функций и структуры программного обеспечения. Очевидно, что модификация поведения программы конечным пользователем возможна только тогда, когда она содержит:

1) некоторое множество параметров, определяющих ее функционирование;

2) доступный для пользователя механизм управления этими параметрами.

Чтобы оценить степень новизны этих предложений, нужно вспомнить определение конечного автомата. *Конечным автоматом A* называют набор

$$A = (X, Q, Y, \alpha, \beta),$$

где X – множество входных сообщений;

Q – множество внутренних состояний автомата;

Y – множество выходных сообщений; $\alpha : X \times Q \to Q$ – функция перевода автомата в новое состояние;

$\beta : X \times Q \to Y$ – функция преобразования входных сообщений в выходные.

Нетрудно видеть аналогии между требованиями к новой структуре программного обеспечения и структурой конечного автомата. Таким образом, разработчики программных средств осознали, что программа должна быть тем, чем она должна быть, – конечным автоматом. Нельзя сказать, что эти идеи овладели массами программистов. По своей структуре большинство современных программ ближе к комбинационной схеме (автомату без памяти), чем к конечному автомату. Управляющие параметры, как правило, размещаются в тексте программы (исходном модуле) и не могут быть изменены без модификации ее исходного текста, недоступного для конечных пользователей.

Реализация принципов новой информационной технологии на практике ведет к повышению адаптируемости программного обеспечения. Поскольку адаптирующиеся системы обладают большей живучестью при изменении внешних условий, постольку основная идея новой информационной технологии – необходимость предвидения будущих изменений и включения в

программное обеспечение средств адекватного реагирования на них – является весьма привлекательной для конечных пользователей.

Новая информационная технология может представлять значительный интерес и для разработчиков, так как адаптируемость в данном случае может служить средством для достижения другой цели – обеспечения универсальности программного обеспечения, способности покрывать широкую область потребностей и, как следствие, увеличения тиражируемости программного продукта. Так как с увеличением размеров систем (в нашем случае под размером системы следует понимать не длину программного кода, а ее функциональные возможности) их количество в замкнутом пространстве убывает, то адаптирующаяся информационная система имеет хорошие шансы на захват значительного сектора рынка.

Одним из направлений в новых информационных технологиях является разработка систем, основанных на знаниях. Надежды, которые связываются с разработкой таких систем, обусловлены тем, что они, во-первых, решают более широкий класс задач, чем традиционные программные системы, а во-вторых, дают пользователю возможность более высокой адаптации к его условиям. Системы, основанные на знаниях, будут рассматриваться в следующих главах.

4.18. Проблема концептуального единства

В настоящее время сколько-нибудь значительные системы, в том числе автоматизированные картографические и геоинформационные системы, создаются коллективами специалистов. Но в любом коллективе возникает проблема взаимного понимания, поскольку члены коллектива разработчиков часто отличаются полученной специальностью, уровнем образования (в том числе – самообразования), опытом работы в конкретной проблемной области. Таким образом, проблема взаимопонимания в коллективе разработчиков сводится к проблеме знания. В разработке систем геомоделирования по необходимости должны участвовать как специалисты в области вычислительной техники и информатики, так и специалисты в области картографии и геодезии, и эти области существенно различаются объектом и предметом своих исследований.

Указанная проблема понимания в исследованиях многосубъектных систем (а любой коллектив разработчиков является такой распределенной системой, так как каждый его член выполняет свои функции относительно автономно) получила название проблемы *группового знания*, или *разделенного знания*. Эта проблема возникает при разработке сложных систем, когда каждый участник разработки должен согласовывать свои действия с действиями всех других членов коллектива. Было предложено (Д. Льюис) следующее понимание группового знания: группа имеет общее знание о *X*, если каждый член группы, во-первых, знает, что *X*, и, во-вторых, знает, что каждый знает, что *X*.

Проблема группового знания не ограничивается только сферой разработки, она имеет более широкое распространение. В этой связи можно привести несколько экстравагантный пример, вошедший в «фольклор» распределенных систем [17]. Пусть две дивизии одной армии находятся на противоположных

склонах долины, занятой противником. Соотношение сил таково, что если обе дивизии атакуют противника одновременно, то они выиграют сражение, но атака силами только одной дивизии приведет к поражению, и командиры обеих дивизий знают об этом. Кроме того, общего плана действий нет, и командиры дивизий могут общаться между собой только через посыльных, которым для преодоления (только ночью) занятой противником местности требуется один час. Существует некоторая вероятность того, что посыльный может не добраться до места назначения, но других способов общения нет.

Командир одной из дивизий решает отправить посыльного с предложением атаковать противника утром следующего дня. Вопрос заключается в том, сколько потребуется времени генералам, чтобы согласовать свои действия. Ответ на него парадоксален: при заданных условиях они не договорятся никогда.

Объяснение состоит в следующем [17]. Пусть посланник генерала А прибывает к В с предложением начать атаку на рассвете. Но генерал В атаковать не будет, так как он понимает, что генерал А не знает, что В получил его записку, и поэтому А атаковать не будет. Поэтому В отправляет связного обратно с согласием на совместные действия. Генерал А получает согласие В, но атаковать противника не будет, поскольку В не знает, что А получил его согласие, и В может предполагать, что А не выступит утром. Поэтому А снова посылает связного с сообщением, что он получил согласие В. Но даже если его подчиненный передаст сообщение генералу В, этого опять будет недостаточно. В [17] делается вывод, что сумма подтверждений не гарантирует достижения согласия, поскольку каждый раз существует возможность невыполнения задания связным.

Данная ситуация в терминах знания объясняется возрастанием глубины «генеральских знаний» от «А знает» к «А знает, что В знает, что А знает» и так далее. В [17] утверждается, что общее знание не достижимо ни в системах, где обмен информацией не гарантирован, ни в системах, где коммуникация гарантирована, но имеется неопределенность во времени передачи сообщений.

Пример с генералами может быть оспорен, поскольку им не требуется абсолютное знание (с бесконечной глубиной), а достаточно некоторой конечной глубины, например, как на рис. 4.14. Но он важен в том смысле, что демонстрирует сложность проблемы коллективного знания. В процессе дискуссий о структуре и функциях разрабатываемой системы разработчики должны понимать не только суть предложений своих коллег, но и причины, по которым они выдвигаются, то есть групповые знания должны обладать определенной глубиной.

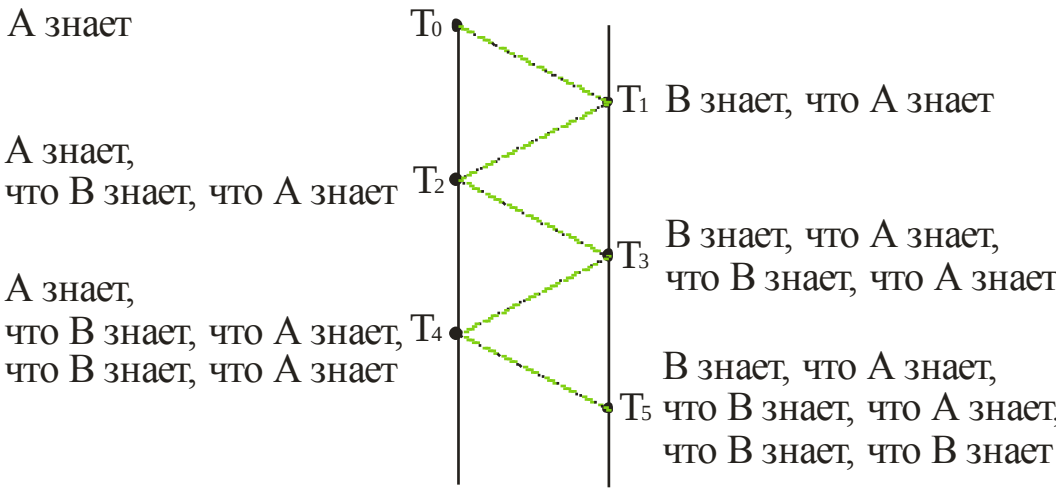


Рис. 4.14. К понятию глубины знаний

Знания рабочей группы не равны сумме знаний ее членов. Это можно показать даже на обыденном уровне. Так, например, если я знаю, что А и В являются сестрами, а вы знаете, что В и С – сестры, то отсюда не следует, что мы с вами знаем, что эти три женщины находятся в определенных родственных отношениях. Мы сможем утверждать, что знаем об этом, только после того, как обменяемся информацией.

Групповое знание оказывает исключительное влияние на качество разрабатываемых систем. Необходимым условием высокого качества разработок является их концептуальное единство, в сущности, это почти синонимы. И концептуальное единство может быть достигнуто только при высоком уровне группового знания. Вообще же концептуальное единство – это такое качество, которого трудно достичь при коллективной разработке. По этой причине одним из авторов была высказана мысль о том, что такие произведения, как «Давид» Микеланджело или «Герника» П. Пикассо, могли зародиться только в одной голове. Это высказывание здесь приводится по той причине, что в произведениях искусства требование концептуального единства столь же важно, как и в технических системах.

Еще одним примером существования проблемы группового знания может служить одна японская техническая выставка, на которой демонстрировался робот, рисовавший портреты посетителей. Интересен не только сам робот, но и отзывы посетителей. Программисты говорили, что им понятно, как робот видит и создает портрет в своей памяти, но удивлялись тому, как он управляет своей рукой. Инженеры-механики понимали, как он манипулирует своими конечностями, но не могли понять, каким образом он создает изображение сидящего перед ним человека. Очевидно, что разработчикам этого робота удалось решить проблему коллективного знания.

Концептуальное единство трудно не только достичь, но и определить. (Возможно, потому и трудно достичь, что трудно определить.) Но, как правило, технические системы, обладающие таким качеством, отличаются удачным сочетанием эстетических и утилитарных функций, то есть тех функций, для выполнения которых они и создаются.

Подтверждением значимости концептуального единства может служить история некоторых разработок Ассоциации по языкам систем обработки данных CODASYL (the **CO**nference on **Da**ta **Sy**stems **L**anguages), созданной в середине 1960-х гг. Работы этой весьма авторитетной организации в области языков программирования и баз данных оказали значительное влияние практически на все аспекты обработки данных. Ассоциацией и ее комитетами была проделана огромная работа в области СУБД сетевой структуры, а некоторые предложенные концепции используются до сих пор. Тем не менее, сетевые СУБД уступили место реляционным, основоположником которых явился Е.Ф. Кодд – математик, работавший в то время в корпорации IBM. Реляционные базы данных будут рассматриваться в следующей главе. Сейчас отметим только, что причиной их массового распространения, по нашему мнению, стало присущее им концептуальное единство, чего нельзя сказать о работах CODASYL. Сетевые СУБД представляются тяжеловесными рядом с реляционными. И, наоборот, теорию реляционных баз данных можно назвать даже изящной, если сравнивать ее с теорией сетевых баз данных.

Сравнение сетевых и реляционных баз данных вызывает в памяти горькую сентенцию о том, что верблюд – это лошадь, созданная комиссией. Сказанное не следует понимать так, будто сетевые СУБД и их теория являются никчемными разработками. Их развитие осуществлялось в течение многих лет самыми квалифицированными специалистами в области обработки данных. Но, в конце концов, лучшее пришло на смену хорошему.

Проблема взаимопонимания в коллективах разработчиков возникает вследствие специализации и дифференциации знаний. Поэтому единственным способом ее решения является следование совету А. Холла, говорившего, что основным занятием системотехника должно быть обильное чтение [24]. К его словам можно добавить еще соблюдение принципов системотехники.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Аветисян Д.А. и др. Системы автоматизированного проектирования: типовые элементы, методы и процессы. – М.: Издательство стандартов, 1985. – 180 с.
2. Большая советская энциклопедия, т. 16. – М.: Советская энциклопедия, 1952.
3. Гвардейцев М.И., Морозов В.П., Розенберг В.Я. Специальное математическое обеспечение управления. – М.: Сов. радио, 1978. – 512 с.
4. Гилой В. Интерактивная машинная графика. – М.: Мир, 1981. – 380 с.
5. Горохов В.Г. Методологический анализ системотехники. – М.: Радио и связь, 1982. – 159 с.
6. Грязнов Б.С. Гносеологические проблемы моделирования // Вопросы философии. – 1967. – № 2.
7. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. – М.: Мир, 1981. – 368 с.
8. Дейкстра Е.В. Дисциплина программирования. – М.: Мир, 1978.
9. Дейт К.Дж. Введение в системы баз данных. – Киев: Диалектика, 1998. – 784 с.
10. Дитрих Я. Проектирование и конструирование: системный подход. – М.: Мир, 1980. – 456 с.
11. Зелковиц М., Шоу А., Гэннон Дж. Принципы разработки программного обеспечения. – М.: Мир, 1982. – 368 с.
12. Касти Дж. Большие системы. Связность, сложность и катастрофы. – М.: Мир, 1982. – 216 с.
13. Клир Дж. Системология. Автоматизация решения системных задач. – М.: Радио и связь, 1990. – 544 с.
14. Корн Г., Корн Т. Справочник по математике для научных работников и инженеров. – М.: Глав. ред. физ.-мат. лит. изд-ва «Наука», 1970. –720 с.
15. Криницкий Н.А., Миронов Г.А., Фролов Г.Д. Программирование и алгоритмические языки. – М.: Глав. ред. физ.-мат. лит. изд-ва «Наука», 1975. – 381 с.
16. Криницкий Н.А., Миронов Г.А., Фролов Г.Д. Автоматизированные информационные системы. – М.: Глав. ред. физ.-мат. лит. изд-ва «Наука», 1982. – 381 с.
17. Логика и компьютер. Моделирование рассуждений и проверка правильности программ / Н.А. Алешина и др. – М.: Наука, 1990. – 240 с.
18. Математический энциклопедический словарь. – М.: Советская энциклопедия, 1988. – 847 с.
19. Основы современной системотехники / Под ред. М. Рабина. – М.: Мир, 1975. – 527 с.
20. Першиков В.И., Савинков В.М. Толковый словарь по информатике. – М.: Финансы и статистика, 1991. – 543 с.
21. Турский В. Методология программирования. – М.: Мир, 1981. – 264 с.

22. Философский словарь / Под ред. И.Т. Фролова. – М.: Политиздат, 1981. – 445 с.
23. Флейшман Б.С. Основы системологии. – М.: Радио и связь, 1982. – 368 с.
24. Холл А. Опыт методологии для системотехники. – М.: Сов. радио, 1975. – 448 с.
25. Холл П. Вычислительные структуры. Введение в нечисленное программирование. – М.: Мир, 1978. – 214 с.
26. Шрейдер Ю.А., Шаров А.А. Системы и модели. – М.: Радио и связь, 1982. – 152 с.
27. Фигуровская В.М. Техническое знание. Особенности возникновения и функционирования. – Новосибирск: Наука, 1979. – 192 с.
28. Дрейфус М., Ганглоф К. Практика программирования на Фортране. – М.: Мир, 1978. – 224 с.

5. СИСТЕМЫ ОБРАБОТКИ ДАННЫХ

5.1. Данные как ресурсы

В процессе своей деятельности любое предприятие использует определенные виды ресурсов: материальные, финансовые, трудовые и т. п. Аналогичным образом, для эффективного управления предприятием требуется разнообразная информация. С течением времени, когда для обработки данных стали применяться ЭВМ, было осознано, что данные также являются ресурсом, и возникло отношение к данным как к ресурсу [4].

Данные обладают определенными свойствами, присущими всем другим видам ресурсов, в частности, они могут характеризоваться:

– *стоимостью*, поскольку для их создания, хранения и обработки требуются те или иные затраты;

– *полезностью*, так как использование данных влияет на эффективность принимаемых решений.

Эффективное использование ресурсов любого типа предполагает управление ими: планирование, учет, поддержание в требуемом виде, сохранение, распределение и использование. Для управления различными видами ресурсов на предприятиях и в организациях существуют соответствующие службы, отвечающие за конкретный вид ресурсов. Так, денежные ресурсы находятся в ведении главного бухгалтера, трудовые ресурсы – в ведении отдела кадров. Материальными ресурсами может управлять отдел снабжения или управляющий по производству, который должен знать о запасах материальных ресурсов, о потребности в них на каждом этапе, производственных процессах, поставщиках, ценах и т. п.

Таким образом, общим подходом к управлению различными видами ресурсов является его (управления) *дифференциация* и *централизация*. Понимание данных как ресурса привело к возникновению *концепции централизованного управления данными*. Проблема централизованного управления данными имеет два аспекта: организационный и технологический. В качестве административного решения задачи получения и использования данных на предприятиях и в организациях создавались специализированные информационные подразделения или службы с передачей им всех функций по управлению данными.

Однако, со временем было также установлено, что одних только административных или организационных мероприятий по созданию и управлению данными недостаточно для их эффективного использования и требуются изменения в *технологиях управления данными*.

Решение разнообразных задач требует различных сведений об одних и тех же объектах, лежащих в сфере интересов информационной системы. Особенностью первых этапов применения ЭВМ для задач обработки данных являлось создание для каждой решаемой задачи специализированных наборов данных, содержащих все необходимые сведения. Управление данными при этом осуществлялось с помощью так называемых *файловых систем*. Результатом такой методологии являлось неизбежное дублирование одних и тех же сведений

в разных наборах данных, что, как правило, требовало значительных усилий со стороны информационных служб по поддержанию данных с целью обеспечения их актуальности и непротиворечивости. Потребности развития технологий управления данными естественным образом привели к осознанию необходимости *интеграции данных* и возникновению *концепции баз данных*.

5.2. Концепция баз данных

На первых этапах использования ЭВМ для обработки данных требования пользователей были довольно простыми, а решаемые задачи – немногочисленными. Данные хранились на внешних носителях данных, а их описание хранилось в документации либо «в голове пользователя» и включалось в программы в неявном (опосредованном) виде. Подобную организацию данных принято называть *файловой.* Для каждой программы данные подготавливались индивидуально, в результате чего данные:

− во-первых, многократно дублировались, что влекло за собой проблему их синхронного обновления во множестве файлов с целью достижения непротиворечивости их содержания;

− во-вторых, оказывались жестко связанными с программами, в результате чего при изменениях в структуре данных требовались значительные затраты на модификацию программного обеспечения.

По мере увеличения вычислительной мощности ЭВМ и расширения круга решаемых задач наряду с эффективностью обработки данных все большее значение стало приобретать требование гибкости данных, возможности их использования в самых разнообразных приложениях. Как реакция на указанные выше недостатки файловой организации данных, возникла идея разделения данных и программ. Новая организация данных, возникшая в 1960-х гг. и основанная на этой идее, получила название баз данных (БД). Для пользователей отличие баз данных от файловой организации заключается в том, что базы данных создаются не для отдельных приложений, а для решения всего множества задач конкретного учреждения или предприятия.

Но принципиальное отличие состоит в том, что на смену сочетанию «данные + программа» при файловой организации пришло сочетание «данные + описание данных + механизм доступа к данным + программы», используемое при организации баз данных.

Базой данных называют структурированную совокупность данных, описывающих состояние конкретной предметной области с полнотой, достаточной для решения всего множества задач.

В первом приближении базу данных можно рассматривать как некоторую электронную картотеку, набор сведений о предметной области. Как и в обычной картотеке, в нее можно добавлять пустые записи (аналог чистых бланков), добавлять новые данные, читать, изменять или удалять уже существующие в ней сведения.

Преимущества баз данных перед обычными картотеками достаточно очевидны:

1) компактность (для сравнения скажем, что Большая Советская Энциклопедия, представляющая собой 30 томов большого формата, размещается на трех оптических дисках емкостью 700 Мб каждый и может быть записана на одном DVD, на котором займет менее 50 % от его объема);

2) скорость обработки (в отдельных случаях она может быть выше скорости ручной обработки на несколько порядков);

3) низкие трудозатраты (при поиске информации нет необходимости в ручной работе с картотекой, достаточно сформулировать запрос, что выполняется намного быстрее);

4) доступность (базы данных могут находиться на значительном удалении от места запроса, тогда как к обычным картотекам нужен непосредственный доступ).

Отмеченные преимущества особенно наглядно проявляются в многопользовательских системах, когда доступ к одной базе данных имеет множество людей, часть которых может находиться на большом расстоянии от нее.

Другими значимыми преимуществами централизованного подхода к управлению данными считаются:

- возможность сокращения избыточности данных;
- возможность избавления от противоречивости и сохранения целостности данных;
- возможность организации общего доступа к данным;
- возможность унификации представления данных;
- возможность защиты данных от несанкционированного доступа;
- возможность оптимизации представления данных.

База данных наряду с собственно данными содержит *описание данных*. *Системой управления базами данных* (СУБД) называют совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования данных различными пользователями. *Банк данных* рассматривается как совокупность программных, лингвистических, технических и организационных средств, предназначенных для централизованного накопления и коллективного использования данных [6] (рис. 5.1). Часть программ СУБД, выполняющих наиболее важные функции и постоянно находящихся в оперативной памяти во время ее работы, образует *резидентный модуль*.

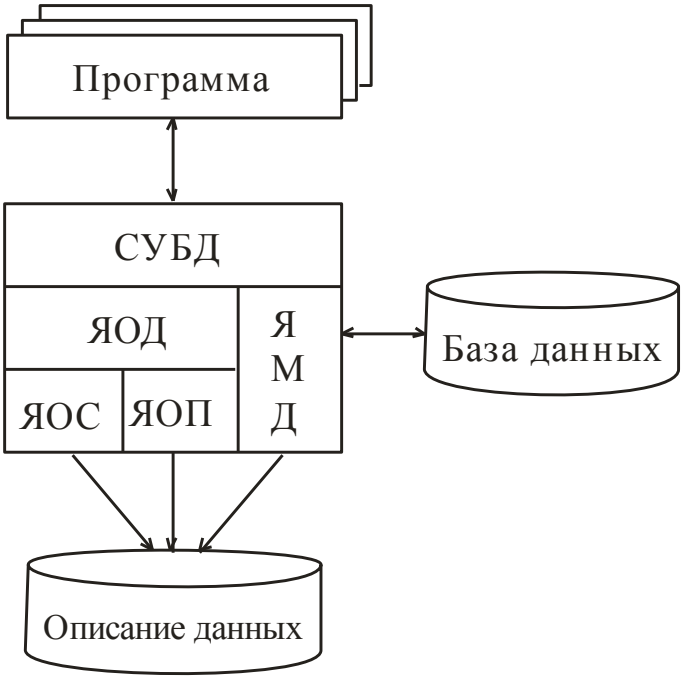


Рис. 5.1. Архитектура СУБД

Языковые средства СУБД подразделяются на язык описания данных и язык манипулирования данными. Для описания структуры базы данных в целом и ее отдельных элементов используется *язык описания схемы* базы данных (ЯОС). Для описания данных, необходимых конкретным программам, служит *язык описания подсхем* (ЯОП). Язык описания схем и язык описания подсхем данных в совокупности образуют *язык описания данных* (ЯОД), являющийся языком декларативного типа. С его помощью какие-либо действия с базой данных не осуществляются.

Для работы с базой данных, ее создания, добавления в нее данных, их изменения или удаления предназначен *язык манипулирования данными* (ЯМД). Любые действия, описываемые на ЯМД и выполняемые над базой данных, в том числе операции чтения данных, называют *транзакциями*.

Интеграция всех необходимых данных в одной базе данных имеет своей целью исключение избыточности данных и требует централизации управления ими. Объектами управления при этом являются не значения самих данных, а структура и содержание (смысл) данных. Функцию управления структурой и содержанием базы данных называют *администрированием данных*.

Дальнейшее развитие баз данных привело к возникновению понятия *метаданных*, под которыми понимаются данные о данных. Метаданные содержат сведения о семантике (смысле) данных, об их технических характеристиках и использовании. Для создания и поддержания метаданных служат словари данных и справочники данных. *Словари данных* содержат в качестве своих статей описания всех используемых элементов и структур данных. Словари данных преимущественно предназначены для конечных пользователей и программистов. *Справочник данных* содержит описание физических свойств данных: способы их представления, размещения и пути доступа к ним. Справочники данных используются в основном программами и процессами, осуществляющими непосредственный физический доступ к данным.

К информационным системам, основанным на базах данных, предъявляются следующие требования:

- интегрированность и возможность совместного использования данных;

– оптимизация использования данных с помощью их соответствующего структурирования;

– независимость данных от приложений (конкретных программ) на основе унификации представления данных;

– гибкость и адаптируемость структуры БД;

– адекватность БД предметной области;

– исключение противоречивости данных и обеспечение их целостности с помощью средств автоматического контроля данных;

– минимизация избыточности данных;

– простота физической реорганизации данных;

– динамичность данных;

– расширяемость состава данных;

– возможность поиска данных по различному содержанию;

– ограничение доступа к данным, защита данных от несанкционированного доступа или модификации.

Наиболее важными свойствами систем обработки данных на основе БД считаются следующие.

Экономия памяти достигается блокированием нескольких логических записей в одну физическую и преобразованием данных в более компактную форму, в том числе их сжатием.

Эффективность обработки данных обеспечивается путем выбора из двух возможных стратегий: последовательной и произвольной. Это объясняется тем, что существуют как задачи, требующие последовательного просмотра всех данных, так и задачи, для решения которых достаточно доступа лишь к некоторому подмножеству данных в произвольном порядке.

Частота обращений. Наиболее часто используемые файлы или их отдельные записи должны располагаться так, чтобы доступ к ним осуществлялся легко и с наибольшей скоростью.

Время ответа. Существуют ситуации, когда время ответа не должно превышать 1–2 с (например, при работе пользователя в интерактивном режиме), но есть и такие, когда оно может составлять 1 час и более. Время ответа должно согласовываться с частотой обращений к данным.

Пропускная способность. Данный показатель имеет важное значение для систем, работающих в режиме реального времени. Поэтому при их реализации должна обеспечиваться возможность *параллельного доступа* к данным.

Динамичность данных. Одни данные могут не изменять своих значений годами, другие же изменяются очень быстро. Примером редко изменяемых данных могут служить данные о ситуации и рельефе в некоторой ГИС. Пример быстро изменяющихся данных – данные о положении подвижных объектов, получаемые с помощью GPS и отображаемые в какой-либо диспетчерской системе. Файлы, записи которых подвержены частым изменениям, называют *динамическими.*

Добавление групп записей. Одновременное добавление большого количества данных требует значительных затрат времени, для уменьшения которых требуется тщательная проработка физической организации данных.

Способность к расширению вызвана тем обстоятельством, что в некоторые файлы данные могут добавляться очень часто, а удаляться из них – крайне редко. В таких случаях необходимо, чтобы соответствующие файлы не имели ограничений на число записей, число используемых томов данных и т. п.

Отсутствие *избыточности данных* достигается тщательным проектированием структуры базы данных и интенсивным использованием указателей.

Активность файла A характеризуется величиной

$$A = \frac{m}{n},$$

где m – число записей файла, считанных и использованных за один его просмотр;

n – число всех записей файла, считанных за один его просмотр.

5.3. База данных как компонент информационной системы

Любая информационная система соотносится с определенной предметной областью – выделенной частью реального мира, представляющей интерес для конкретного множества пользователей. Состояние базы данных пользователями трактуется как отображение состояния предметной области, а любые изменения в ней – как изменения объективной действительности. Поэтому конкретную базу данных принято рассматривать как *информационную модель* определенной предметной области, эволюционирующей во времени.

Информационные системы необходимо рассматривать в двух аспектах. Первый из них связан со способностью информационной системы накапливать, хранить, корректировать, извлекать и обобщать сведения о выделенных фрагментах реального мира, являющихся предметом моделирования. Второй аспект информационной системы связан с возможностью использования данных об объективной действительности для получения эффективных решений задач в конкретной проблемной области.

Основным свойством информационной системы является то, что для принятия обоснованных решений или оказания воздействия на предметную область требуется извлечение информации из ее модели (рис. 5.2). Если база данных содержит достаточные и достоверные сведения о предметной области, то она может быть использована для прогнозирования последствий предполагаемых действий.

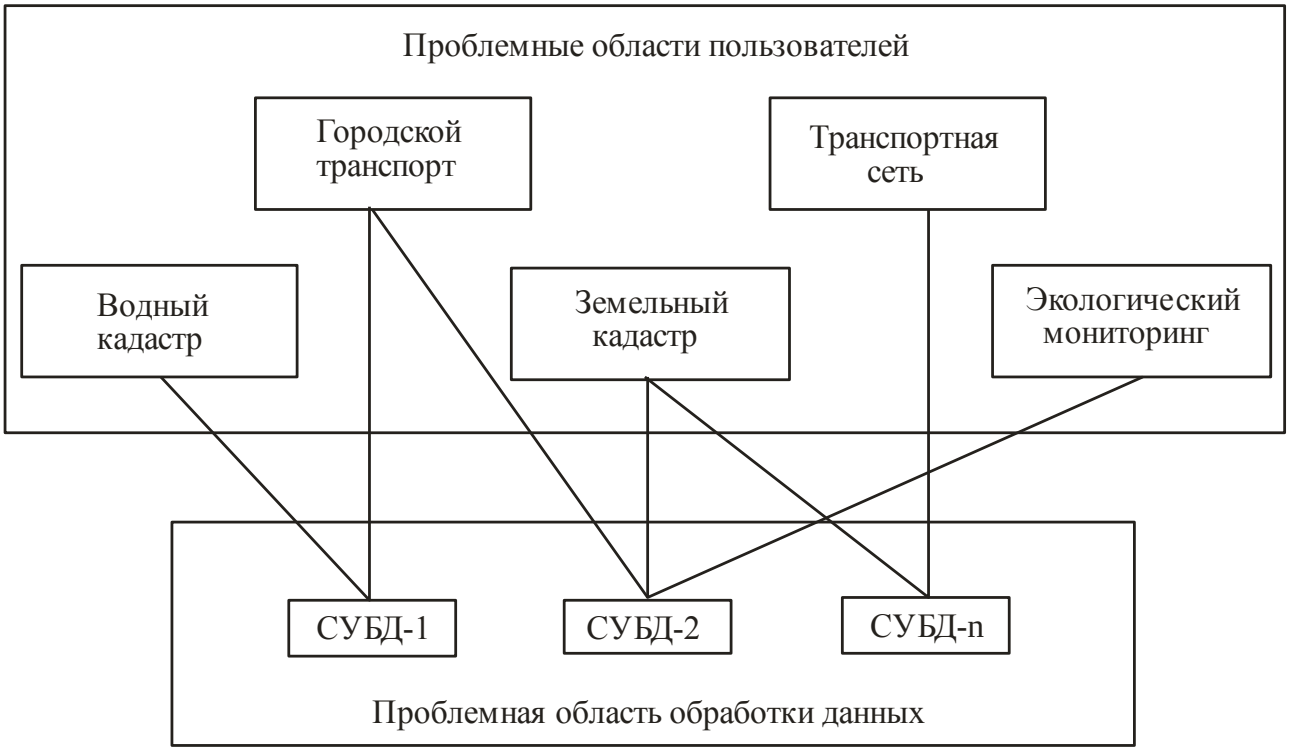


Рис. 5.2. Проблемные области

Информационная система может мыслиться как совокупность определенных функций, для выполнения которых она и создается. Каждая такая функция реализуется как одна или несколько взаимосвязанных программ, образующих программную систему. В процессе функционирования каждой программной системы ей требуются сведения о некоторой части предметной области. Фрагменты предметной области, необходимые разным программным системам, могут полностью или частично совпадать, пересекаться. Но даже в тех случаях, когда используются данные об одних и те же объектах, видение этих объектов в разных программах, как правило, различно. В таких случаях говорят, что данные о реальных объектах имеют различное *представление*, под которым понимают форму выражения данных о предметной области.

Таким образом, каждая база данных создается, с одной стороны, с целью удовлетворения всех информационных потребностей в конкретной проблемной сфере, а с другой – с целью обеспечения эффективного представления данных для решения каждой из множества частных задач. В разрешении этого противоречия заключается основная сложность разработки структуры баз данных.

База данных содержит лишь те сведения о предметной области, которые представляют интерес для информационной системы. Следовательно, в течение всего времени существования информационной системы необходимо уделять достаточно внимания полноте, достоверности и актуальности данных о предметной области. При этом предполагается, что каждому состоянию предметной области соответствует определенное состояние (содержание) базы данных. Все множество возможных состояний предметной области должно быть представимо в базе данных. Такое множество называют *множеством воображаемых состояний предметной области*. Кроме того, предполагается,

что различные категории пользователей базы данных имеют согласованные представления об отображаемой в ней предметной области и в любой момент времени придерживаются единого мнения относительно соответствия между содержимым БД и предметной областью [3].

Состояние предметной области считается *определенным*, если известны все объекты, их свойства и межобъектные отношения. Описание состояния предметной области представляется с помощью последовательностей символов, то есть некоторых языковых средств. Отсюда следует, что все пользователи базы данных должны понимать используемый язык.

Для человека наиболее доступным является описание предметной области на естественном языке. Но описание состояния предметной области в базе данных представлено на формальном языке. Поэтому соотнесение представления предметной области в базе данных с ее реальным состоянием осуществляется только через ее описание на естественном языке. Отсюда следует необходимость перевода описания на одном языке в описание на другом.

Существует множество предметных областей, для отображения которых могут создаваться информационные системы на основе баз данных. С другой стороны, имеется достаточно большое количество СУБД, каждая из которых может использоваться для моделирования различных предметных областей (см. рис. 5.2). При этом разработка методов представления и обработки данных в любой проблемной области представляет самостоятельную проблемную область.

В данной главе не рассматриваются проблемные области конечных пользователей, например, такие как градостроительный кадастр и другие, в том числе перечисленные на рис. 5.2. Предметом нашего рассмотрения являются общие принципы создания баз данных и информационных систем на их основе.

Использование того или иного языка для описания предметной области основывается на некотором наборе понятий. Ниже рассматриваются понятия, используемые при описании такой проблемной области, как разработка или анализ различных баз данных и СУБД. Объектами, изучаемыми в рамках соответствующей теории, являются модели данных.

Общая схема использования баз данных в автоматизированных информационных системах представлена на рис. 5.3.

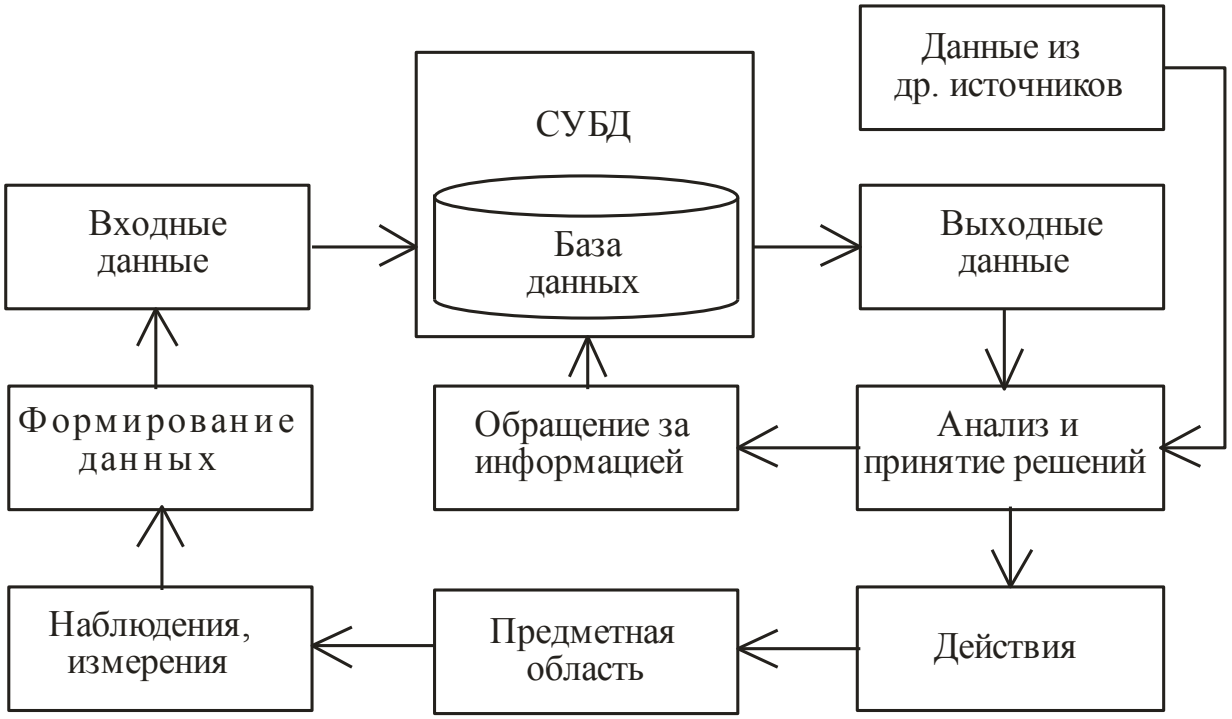


Рис. 5.3. База данных в АИС

5.4. Уровни представления данных

Представлением данных называют совокупность правил кодирования элементарных данных и правил образования из них более сложных конструкций [5]. Данные о конкретной предметной области могут рассматриваться с различных позиций. Но таким же образом данные о разных предметных областях могут рассматриваться с одной и той же точки зрения. При определении содержания и структуры баз данных принято выделять *три уровня рассмотрения*, которые называют концептуальным, внутренним и внешним представлениями (рис. 5.4). Наличие трех указанных уровней представления данных объясняется условиями использования данных.

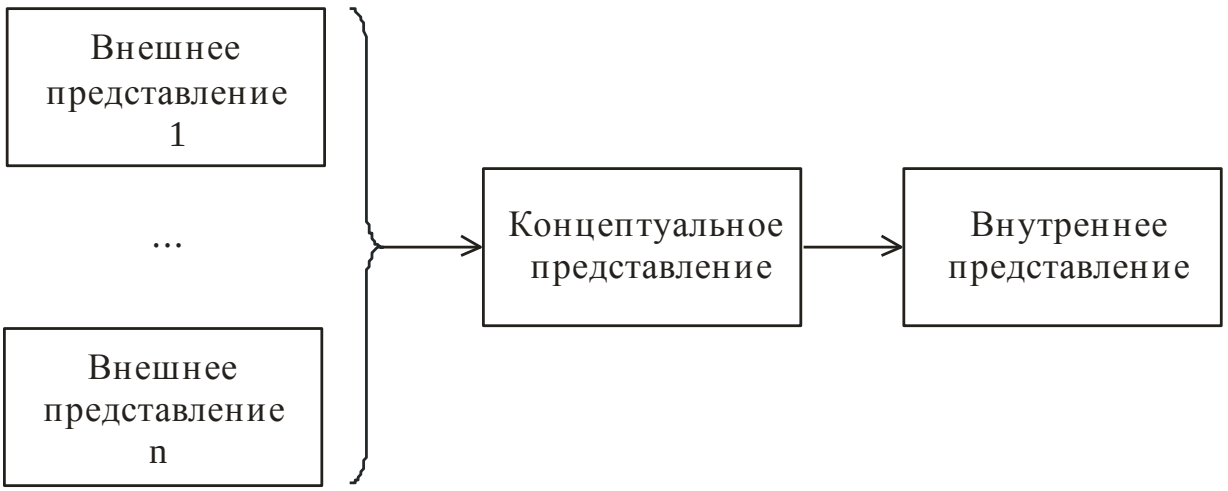


Рис. 5.4. Уровни представления данных

В каждой проблемной области существует множество задач, которые должны решаться с той или иной периодичностью. Некоторые из этих задач

могут частично или полностью перекрываться по данным с другими задачами. *Концептуальное представление* данных является объединением всех данных, необходимых для решения всего спектра задач в проблемной области. Иначе, на *концептуальном уровне* данные рассматриваются и описываются с точки зрения всей информационной системы в целом. Элементарными единицами концептуального представления являются *элементы* предметной области (сущности, объекты, процессы…), *свойства* элементов и *связи* между элементами.

Для решения конкретного класса задач требуется, как правило, лишь некоторое подмножество данных. Множество данных, необходимых для решения частных задач, называют *внешним представлением* данных, или *внешним уровнем*. Этот уровень выделяется в связи с тем, что программисту при разработке программы для решения вполне определенных задач не требуется знания всей совокупности данных, так же как не требуется знания деталей их внутренней организации. По этой причине внешний уровень данных называют также *логическим*.

Таким образом, концептуальное представление данных является полным описанием информационных потребностей, получаемым путем интеграции всех внешних представлений проблемной области (см. рис. 5.4).

Преимущество информационных систем, созданных на основе СУБД, – в том, что способы организации данных и методы доступа к данным оказываются скрытыми от прикладного программиста и пользователя. Вследствие этого они получают возможность сосредоточиться на решении задач, а не на разработке или изучении способов организации данных. Кроме того, использование СУБД позволяет существенно ослабить зависимость прикладных программ (приложений) от способов организации данных.

Представление и описание данных, рассматриваемое с точки зрения организации их физического размещения и хранения на носителях информации, называют *внутренним уровнем*, или *внутренним представлением* данных. Внутренний уровень данных называют иногда *физическим* уровнем.

Внутренняя запись является хранимой записью. Внутренний уровень наиболее близок к физическому уровню, то есть уровню, с которым непосредственно работает система управления базой данных.

Это представление называют внутренним по той причине, что оно скрыто от пользователей и прикладных программистов, занимающихся решением конкретных задач. Внутреннее представление данных – это представление, с которым непосредственно взаимодействует СУБД, не выполняющая содержательной обработки данных. Содержательная обработка данных в базе данных выполняется прикладными программами.

Основными компонентами внутреннего представления являются *физические записи*, иногда называемые также *физическими блоками*, указатели, данные переполнения и межблочные промежутки. Блокирование нескольких логических записей в одну физическую не оказывает влияния на их содержательную обработку, но позволяет повысить ее эффективность. Число логических записей в одной физической записи называют *коэффициентом*

блокирования. Именованную целостную совокупность физических записей называют *файлом*, или *набором данных*. Но иногда под файлом понимают и совокупность логических записей.

С внутренним представлением данных связан *метод доступа* – совокупность правил и средств, с помощью которых осуществляется непосредственный доступ к физическим записям наборов данных. В зависимости от организации данных различают три основных метода доступа: *последовательный*, *прямой* и *индексный*.

5.5. Структурированные и неструктурированные данные

Все множество баз данных может быть разбито на два самых крупных класса: базы структурированных данных и базы неструктурированных данных [3]. *Базы структурированных данных* характеризуются тем свойством, что на стадии разработки структуры каждой такой базы фиксируются:

1) все возможные *типы* объектов;
2) все типы *свойств*, присущих каждому типу объектов;
3) все возможные *отношения* между типами объектов.

Таким образом, схема (организация) базы структурированных данных представляет собой множество утверждений о возможных состояниях предметной области в виде фактов, имеющих заранее определенную совокупность форматов. Следовательно, базы структурированных данных требуют полной предварительной структуризации предметной области и фиксирования допустимых форматов. Каждый такой формат может быть описан соответствующим предложением естественного языка. Иногда базы структурированных данных называют также *форматированными базами данных* и *базами данных с детерминированной схемой*.

Базы структурированных данных используются для описания массовых свойств и отношений в предметной области. Так, например, в системе геомоделирования может быть несколько десятков или сотен *типов объектов*, а *число объектов* некоторых типов может составлять десятки и сотни тысяч (здания, горизонтали, значения высот и т. п.).

Базы структурированных данных получили наибольшее распространение. Однако существуют предметные области, где свойства и отношения не имеют такой массовости и характеризуются большим разнообразием и индивидуальностью. Примерами подобных областей могут служить диалоговые человеко-машинные системы, анализ текстов на естественных языках. В базах данных для таких приложений совокупность свойств объектов и отношений определяется только в момент появления нового типа объекта в базе данных. В СУБД, поддерживающих подобные базы данных, могут отображаться факты, для описания которых требуется использование предложений естественного языка с незаданной заранее структурой. По этой причине указанные базы данных называют *базами неструктурированных данных*, *неформатированными базами данных* и *базами данных, не имеющими детерминированной схемы*.

Существуют СУБД, в которых наряду с заранее определенными типами объектов, их свойств и отношений допускается доопределение произвольных

свойств и отношений в момент ввода новых фактов в базу данных. Базы данных подобного типа называют *смешанными,* или *базами данных с частично-детерминированной структурой*.

5.6. Базисный набор типов структур данных

Базы структурированных данных отличаются значительным разнообразием. Для описания их структуры и состояния может быть использована достаточно общая неформальная система понятий, рассматриваемая ниже.

По своей структуре данные подразделяются на элементарные и составные [3]. Под э*лементарными данными* понимается минимальная (неразлагаемая) часть данных, имеющая смысл. *Составными данными* называют некоторую совокупность (конструкцию), в которую могут входить как элементарные, так и другие, менее сложные, составные данные.

С общей точки зрения можно считать, что база данных представляет собой определенным образом организованную совокупность данных, конструируемую из структур данных различных типов. Основное свойство структур данных состоит в том, что одни структуры данных могут создаваться из других, более простых типов структур данных. Из двух структур данных включающая структура имеет *более высокий уровень* и считается *старшей*, а включаемая в нее находится на *более низком уровне* и называется *младшей*.

Описание структуры данных на формализованном языке называют *схемой структуры данных*. Схема определяет структуру данных как целостное образование. *Реализацией схемы* называют конкретную структуру данных соответствующего типа. Правила образования схемы и правила создания реализаций схемы устанавливаются используемым языком определения данных.

По мере возрастания уровня в базисном наборе типов структур выделяются следующие: элемент данных, группа, групповое отношение и база данных.

Элементы данных являются атомарными, нерасчленяемыми структурами данных. Каждый элемент данных имеет уникальное имя, по которому схема и реализация элемента однозначно идентифицируются в схеме и реализации включающей структуры данных. Основными элементами данных являются элементы скалярного типа. *Элементы скалярного типа* характеризуются множеством допустимых атомарных значений и могут определяться либо *перечислением* всех допустимых значений, либо *указанием имени типа* (символьный, целый, вещественный, …), либо *с помощью шаблона*, задающего допустимое значение в каждой позиции.

Элементы скалярного типа подразделяются на *дискретные типы* и *вещественный*. К дискретным типам относятся *перечисляемые типы*, *целый тип* и *логический тип*. Наряду с этим, скалярные типы подразделяются на *числовые* (целые и вещественные) и *нечисловые* (символьные и логические). Каждый скалярный тип доопределяется еще одним особым значением, которое трактуется как *неопределенное* или *неизвестное* значение элемента данных.

Использование данного значения позволяет хранить в базе данных неполные сведения о предметной области. Такие неопределенные значения в базе данных могут в дальнейшем заменяться действительными значениями по мере их выяснения.

В отличие от скалярных типов, *многозначные типы*, называемые *векторными*, при их реализации могут принимать несколько значений (*вектор значений*). Примером значений векторного типа может служить порода деревьев в лесу (когда таких пород несколько).

Для обозначения в некоторой структуре других структур и описания любых отношений между структурами данных вводится специальный тип элементов, называемый *ссылочным*. Данные ссылочного типа можно считать *указателями*, то есть данными, значения которых являются адресами других данных. Подобную адресацию иногда называют *косвенной*.

Группа является линейной последовательностью элементов данных и/или других групп и представляет обычно некоторый тип объектов из предметной области. Группа, включающая только элементы данных, называется *простой*, а если в ее состав входят и другие группы, то *составной*. Схема группы определяет тип группы, содержит несколько схем элементов данных и групп и связывает с каждой группой некоторое уникальное *имя группы*. Реализация схемы простой группы содержит реализацию схемы каждого включаемого элемента данных. Реализация составной группы содержит реализацию схем всех включаемых элементов данных и групп, включая все вложенные группы.

Вложенные группы могут быть повторяющимися и неповторяющимися. *Неповторяющаяся группа* допускает только одну реализацию ее схемы в схеме составной группы. *Повторяющаяся* (или *периодическая) группа* предполагает возможность включения нескольких ее схем в схему составной группы. Множество реализаций схемы повторяющейся группы называют также *ансамблем*, а отдельные реализации схемы повторяющейся группы, входящие в ансамбль, – *членами ансамбля*. Один или несколько элементов данных, однозначно идентифицирующих любой член ансамбля, называют *идентификатором группы.*

Групповым отношением называют бинарное отношение s_k между двумя множествами $s_k \subseteq G_i \times G_j$. Групповое отношение позволяет устанавливать различные отношения между группами (объектами). Групповое отношение характеризуется *коэффициентом группового отношения*, который иногда называют также *ассоциативностью* и который может принимать значения:

– $1:1$, если между элементами множеств G_i и G_j существует взаимно однозначное соответствие;

– $1:n$, если любому элементу множества G_i может быть поставлено в соответствие несколько элементов множества G_j , но каждому элементу множества G_j соответствует только один элемент множества G_i;

– $n:1$, если каждому элементу множества G_j ставится в соответствие несколько элементов множества G_i , но любому элементу множества G_i соответствует только один элемент множества G_j;

– $m:n$, если любому элементу множества G_i соответствует несколько элементов множества G_j и наоборот.

Примером коэффициента группового отношения может служить тип брака. Пусть G_i – множество мужчин, а G_j – множество женщин. Тогда христианский брак характеризуется коэффициентом группового отношения $1:1$, а мусульманский коэффициентом $1:n$. В некоторых племенах, где до наших времен сохранились отношения матриархата и одна женщина может иметь нескольких мужей, коэффициент группового отношения равен $n:1$. В племенах же, где до сих пор существует полигамия, брачные отношения характеризуются коэффициентом группового отношения $m:n$.

В отношениях с коэффициентами $1:1$, $1:n$ и $m:n$ группу g_i называют *родительской группой*, а группу g_j – *подчиненной*, а в отношении с коэффициентом $n:1$ – наоборот. Отношения вида $1:n$ возникают в тех случаях, когда включаемая группа является повторяющейся.

Групповое отношение, в котором каждая подчиненная группа связана не более чем с одной родительской группой, называют *иерархическим.* Иерархическими являются групповые отношения с коэффициентами $1:1$, $1:n$ и $n:1$. Отношения с коэффициентом $m:n$ называются *неиерархическими*.

По другой классификации групповые отношения $1:1$, $n:1$ и $1:n$ называют *функциональными.* Для отношений вида $1:1$ и $n:1$ это очевидно, а для отношения $1:n$ функциональным является обратное отношение s_k^{-1}.

Для наглядного представления групп и групповых отношений используются диаграммы. Группы на них изображаются прямоугольниками, над левым верхним углом которых указывается *имя группы*. Внутри прямоугольника могут указываться *имена элементов данных*. Групповые отношения изображаются стрелками от родительской группы к подчиненной с возможным указанием коэффициента группового отношения. Отношения вида $1:n$ могут обозначаться двойной стрелкой от родительской группы к подчиненной.

Диаграмму групповых отношений можно рассматривать как ориентированный граф, вершины которого отождествляются с группами, а групповые отношения – с дугами.

В примере групповых отношений на рис. 5.5 предполагается, что одно физическое лицо может быть владельцем нескольких зданий, а одно здание может иметь нескольких владельцев. Поэтому коэффициент группового отношения в данном случае $m:n$. Поскольку угловые здания кварталов обычно имеют два адреса, коэффициент отношения между родительской группой «здание» и подчиненной группой «адрес» равен $1:n$.

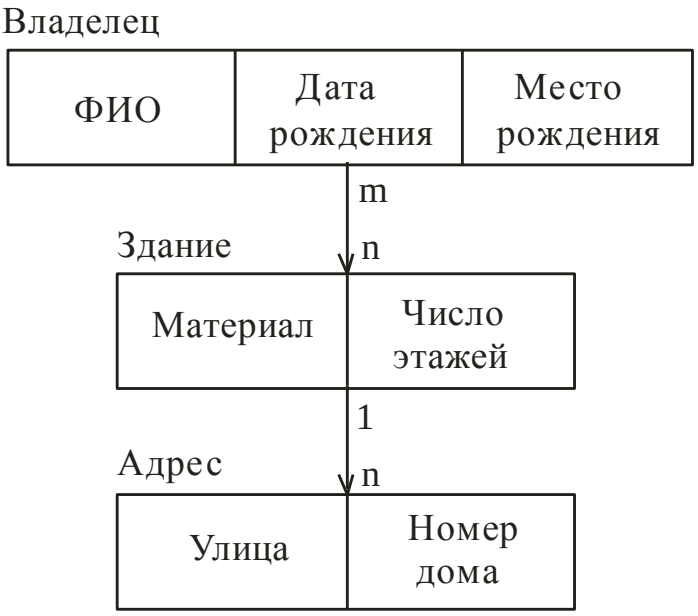


Рис. 5.5. Пример групповых отношений

В базе данных конкретная реализация группы $g_i \in G_i$ может быть идентифицирована либо присваиванием ей уникального имени, либо по ее связям с другими группами в групповых отношениях, либо некоторой неизбыточной совокупностью своих элементов данных K_{g_i}. Группы, идентифицируемые с помощью K_{g_i}, называются *группами с внутренней идентификацией.* На диаграммах группы с внутренней идентификацией обозначаются символом # внутри либо снаружи прямоугольника, а имена элементов данных, входящих во внутренний идентификатор группы, подчеркиваются. Отсутствие символа # означает, что группа не имеет внутренней идентификации.

Псевдоключом подчиненной группы $g_j \in G_j$ в групповом отношении s_k при родительской группе $g_i \in G_i$ называют неизбыточную совокупность элементов данных, входящих в G_j, таких, что конкретный набор их значений позволяет однозначно идентифицировать определенную группу g_i, $(g_i, g_j) \in s_k$, среди всех других групп – образов g_i в отношении s_k. Псевдоключ ставится в соответствие групповому отношению и обозначается как Ps_k.

Селектирующим путем Ws в схеме базы данных называют последовательность типов групповых отношений $Ws = (S_a, S_{a+1}, ..., S_{a+n})$, в которой для $0 \le i \le n-1$ первый домен отношения S_{a+i+1} совпадает с последним доменом отношения S_{a+i}. Необходимо, чтобы первая группа в селектирующем пути $G_j (S_a \subset G_j \times G_k)$ являлась группой с внутренней идентификацией.

Селектирующий ключ родительской группы $g_i \in G_i$ в групповом отношении s_k посредством селектирующего пути $Ws = (S_a, S_{a+1}, ..., S_{a+k-1})$

есть совокупность элементов данных $Cs_k = (K_{g_f}, Ps_a, Ps_{a+1}, ..., Ps_{a+k-1})$, где G_f является первым доменом отношения S_a.

Селектирующим ключом подчиненной группы $g_j \in G_j$ в групповом отношении s_k посредством селектирующего пути $Ws = (S_a, S_{a+1}, ..., S_{a+k})$ называют совокупность элементов данных $Es_k = (Cs_k, Ps_k)$.

Иногда селектирующий ключ родительской группы называют внешним идентификатором подчиненной группы, который вместе с псевдоключом Ps_k образует полный идентификатор подчиненной группы. Существуют подчиненные группы, не имеющие внутренней идентификации, включая псевдоключ, и называемые внутренне неидентифицируемыми. Для идентификации таких групп требуется не менее двух внешних идентификаторов.

Групповое отношение $s_k \subseteq G_i \times G_j$ называют идентифицирующим, если подчиненные группы $g_j \in G_j$ идентифицируются с помощью внешнего идентификатора Cs_k и псевдоключа Ps_k, в противном случае отношение s_k называют неидентифицирующим.

Идентифицирующее групповое отношение принадлежит к типу 1, если коэффициент группового отношения 1 : n, и различным реализациям подчиненных групп с одним и тем же значением псевдоключа соответствуют различные экземпляры объектов предметной области. Пример такого отношения приведен на рис. 5.6.

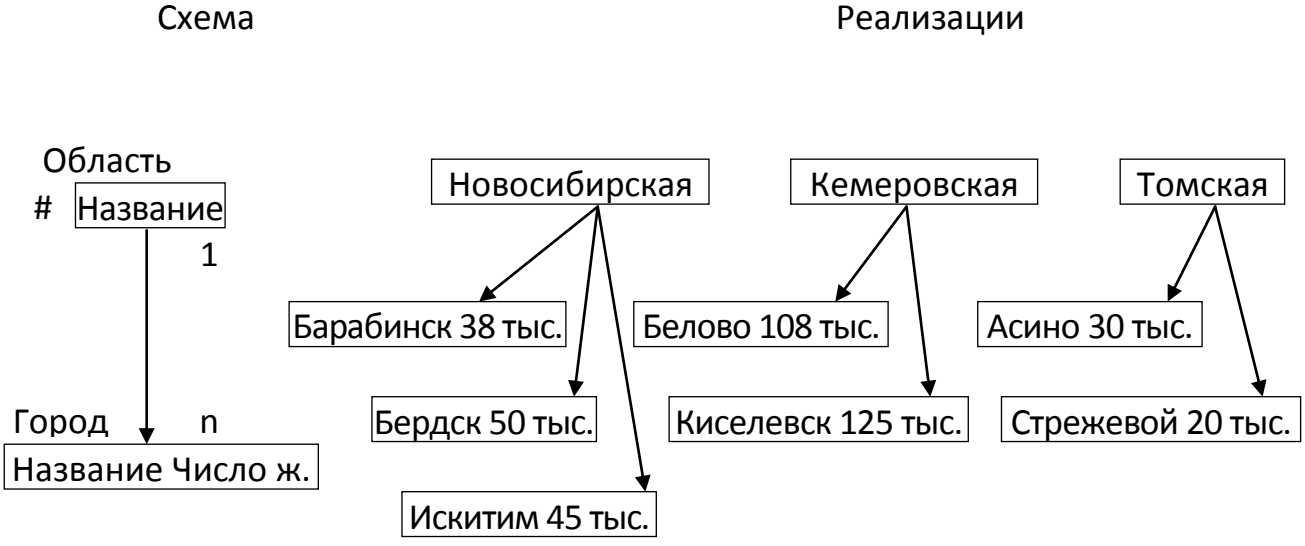


Рис. 5.6. Пример идентифицирующего отношения типа 1

Идентифицирующее групповое отношение принадлежит к типу 2, если коэффициент группового отношения $m : n$, и различным реализациям подчиненных групп с одним и тем же значением псевдоключа соответствуют

одни и те же экземпляры объектов предметной области. Пример идентифицирующего группового отношения приведен на рис. 5.7.

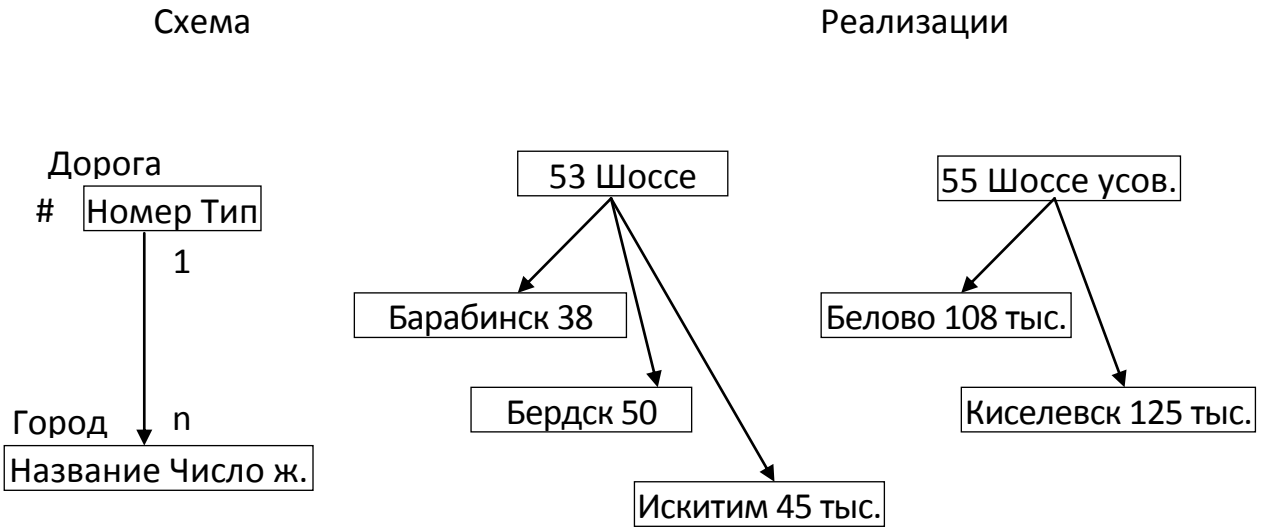


Рис. 5.7. Пример идентифицирующего отношения типа 2

Неидентифицирующее групповое отношение обычно имеет коэффициент $m:n$. На диаграммах такие отношения отображаются двойной стрелкой. Для неидентифицирующего отношения $s_k \subseteq G_i \times G_j$ требуется вводить дополнительные свойства, которые не могут быть включены в схемы групп G_i и G_j. Такие свойства характеризуют пару групп $(g_i, g_j) \in s_k$. Так, пара (пункт отправления, пункт назначения), описывающая междугородный автобусный маршрут, может характеризоваться временем отправления, а пара (источник выброса, вещество) – временем выброса данного вещества в атмосферу.

Каждое неидентифицирующее групповое отношение может быть заменено двумя идентифицирующими отношениями и дополнительным типом группы «связь». На рис. 5.8 показано преобразование схем подобного типа. Понимать его надо так, что имеется некоторое множество предприятий, осуществляющих выбросы вредных веществ в атмосферу (группа «источник»). Причем любое из предприятий может выбрасывать несколько веществ. На рис. 5.8 слева показано, что отношение «источник – вещество» является неидентифицирующим.

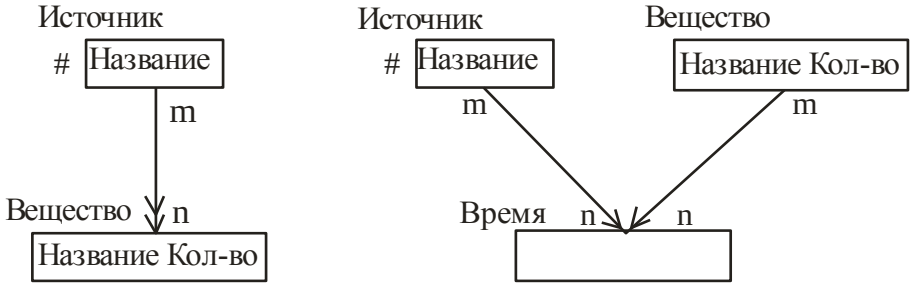


Рис. 5.8. Пример неидентифицирующего отношения

Справа на рис. 5.8 приводятся два идентифицирующих отношения типа 2. Конкретная реализация группы «время» (конкретный выброс фиксированного вещества в определенное время) может быть идентифицирована только двумя внешними идентификаторами, определяющими родительские группы «источник» и «вещество».

5.7. Модели данных

Формализация предметной области порождает необходимость различения двух видов моделей: концептуальной (называемой также инфологической) и даталогической. *Концептуальная модель* представляет собой описание предметной области с точки зрения пользователей. *Даталогическая модель* – это реализация концептуальной модели в вычислительной среде. И даталогическая модель отражает видение данных специалистами по информатике.

Естественно, что применение ЭВМ для моделирования той или иной предметной области начинается на самом первом этапе такого моделирования – формализации, абстрактном описании предметной области. Отсюда следует, что для автоматизированного создания описания предметной области требуются те или иные технологии и инструментальные средства: языковые и программные. Формализованный язык для описания состояния предметной области основывается на некотором наборе используемых первичных понятий. Развитие теории управления базами данных привело к созданию различных языков описания состояния предметной области. Используемый в каждом из них понятийный базис отражает специфику подхода к моделированию предметной области, в том числе – употребляемых типов данных.

Базу данных с даталогической точки зрения можно рассматривать как совокупность групп и групповых отношений. Тогда *схема базы данных* есть объединение схем типов групп и типов групповых отношений. Диаграмму базы данных можно трактовать как ориентированный граф, содержащий несколько компонент связности.

Состояния базы данных интерпретируются как состояния предметной области. Все множество допустимых состояний базы данных определяется ее схемой (структурой) и зависит от возможностей языка определения данных. Перевод базы данных в другое состояние и извлечение данных из БД обеспечивается средствами языка манипулирования данными.

Как правило, используя групповые отношения, нельзя создавать схему базы данных произвольным образом. В рамках каждого используемого для описания данных формализма имеют место те или иные специфические ограничения на создание конструкций, называемые *правилами конструирования схемы*, устанавливающими в первую очередь вид допустимых диаграмм схемы БД. Эти правила отражают понимание и взгляды разработчиков СУБД на проблему разработки и использования баз данных.

Совокупность языка описания данных и языка манипулирования данными определяют *модель данных*, под которой понимают некоторый формализм, использующий фиксированную систему понятий и служащий для отображения

состояния и динамики предметных областей в базах данных. Та или иная модель данных полностью характеризуется языком определения данных и языком манипулирования данными, то есть совокупностью методов и средств определения логической структуры базы данных и динамического отображения состояний предметной области в базе данных.

Согласно ГОСТ 20866–85, *модель данных* определяется как «совокупность правил порождения структур данных в базах данных, операций над ними, а также ограничений целостности, определяющей допустимые связи и значения данных, последовательности их изменения».

Таким образом, *модель данных* может быть определена как абстрактная система, представляющая собой совокупность (единство) трех компонентов:

- набора *типов объектов* данных, из которых может строиться любая база данных, отвечающая требованиям такой модели;
- набора *правил целостности*, ограничивающих множество экземпляров типов объектов, допустимых в подобной базе данных;
- набора *операторов,* которые могут быть применимы для обработки экземпляров объектов.

Назначение типов объектов и операторов достаточно очевидно, назначение же правил целостности состоит в том, чтобы информировать СУБД о разного рода ограничениях, характерных для моделируемой предметной области [1].

Наиболее известными и распространенными являются три типа моделей данных: иерархическая, сетевая и реляционная. Соответственным образом базы данных по своей структуре подразделяются на иерархические, сетевые и реляционные. В соответствии с другой классификацией, БД разделяют на иерархические, неиерархические и смешанные.

Наиболее простыми являются иерархические БД. Каждый компонент связности схемы иерархической базы данных должен обладать следующими свойствами:

- все отношения должны быть идентифицирующими;
- иметь единственный тип группы, не являющийся подчиненным в групповом отношении и называемый *корневым*;
- каждый тип группы, кроме корневого, должен быть подчиненным только в единственном групповом отношении.

При использовании *иерархической модели* данных структура данных имеет вид дерева. Поэтому ее применение наиболее целесообразно для представления отношений, по своей природе являющихся иерархическими. Такими отношениями являются родовидовые отношения и отношения агрегации (рис. 5.9).

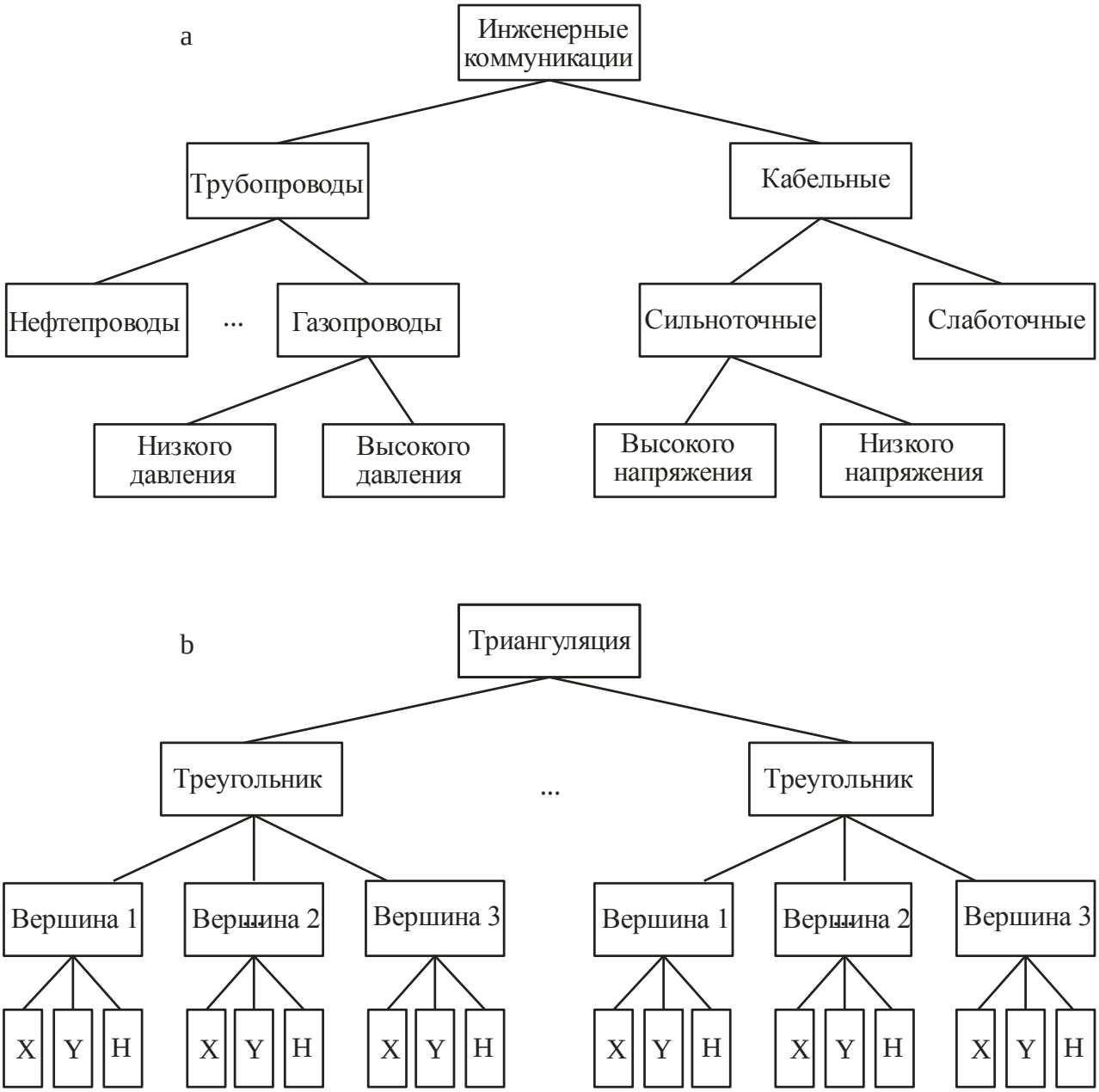


Рис. 5.9. Примеры иерархической структуры

Иерархическая модель представляет данные как иерархию элементов, называемых *узлами*. Каждый узел содержит набор атрибутов, описывающих данный узел. Один из узлов и только один не имеет вышележащих узлов и является *корневым*. Любой не являющийся корневым узел связан только с одним узлом верхнего уровня, называемым *исходным*. Узлы нижнего уровня, связанные с некоторым узлом, называются *порожденными*. Узлы, не имеющие порожденных узлов, называются *листьями*.

Другой пример иерархической базы данных приведен на рис. 5.10. Неиерархические базы данных должны включать неидентифицирующие отношения. В смешанных базах данных допускаются групповые отношения любого вида без ограничений на участие подчиненных групп в отношениях. Сетевые структуры являются наиболее универсальными и в них могут быть

использованы идентифицирующие отношения, неидентифицирующие отношения и смешанные наборы отношений.

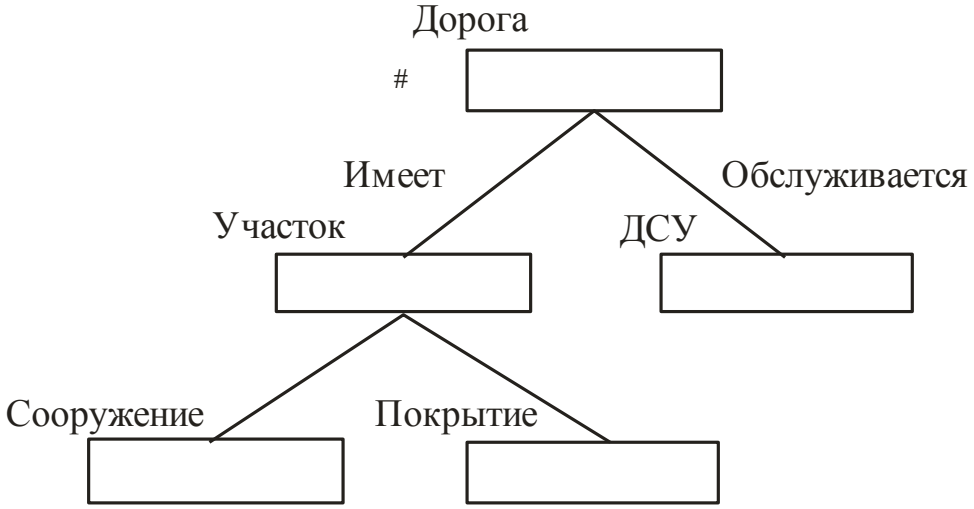


Рис. 5.10. Пример иерархической БД

Концептуальная схема является интегральным определением данных, которое должно строиться так, чтобы представления, заданные различными внешними схемами для всего множества прикладных задач, были эффективно из него выводимы. В рамках концептуальной модели должны решаться такие задачи, как установление непротиворечивости базы данных и управление использованием информации. Поэтому создание концептуальной схемы играет ключевую роль в анализе и проектировании структуры баз данных.

Концептуальной моделью данных определяются следующие важные средства:

1) средства задания пространства данных, используемого для отображения предметной области;

2) средства и методы задания правил, определяющих множество допустимых состояний базы данных, а также правил, определяющих для каждого состояния базы данных множество допустимых состояний, в которые база данных может быть переведена;

3) средства идентификации данных, позволяющие выделять их подмножества, необходимые для решения конкретных задач;

4) средства манипулирования данными, позволяющие привести базу данных в требуемое состояние.

5.8. Отношения в реляционных базах данных

В настоящее время наибольшее распространение получили реляционные БД, основанные на реляционной модели данных. Фундаментальным понятием, используемым в теории реляционных баз данных, является понятие отношения. Отношения при этом рассматриваются как математические объекты. Если даны n множеств $D_1, ..., D_n$ и R есть множество кортежей вида $(d_1, ..., d_n)$ длины n, где $d_i \in D_i \ (i = 1, ..., n)$, то R называют *отношением над этими множествами*, а сами множества $D_1, ..., D_n$ – *доменами*.

Пусть даны представленные на рис. 5.11 множества: *D1* – регистрационные номера продаваемых квартир в некоем агентстве недвижимости, *D2* – множество названий городских улиц, *D3* – множество этажей (конечное подмножество целых чисел), *D4* – материал постройки. Тогда представленный на рис. 5.11 список продаваемых квартир с математической точки зрения можно трактовать как отношение. Каждая строка этого перечня является кортежем. В каждом кортеже на первом месте стоит элемент из первого множества (домена *D1*), на втором – из домена *D2* и т. д. Позицию, которую занимает в отношении домен, называют *атрибутом*. Таким образом, кортеж есть последовательность значений атрибутов.

Число кортежей в отношении называют *мощностью*, или *кардинальным числом*, а число атрибутов в кортеже – *степенью* (или *арностью*) *отношения*. Число атрибутов в отношении, как правило, постоянно; изменяться оно может только при изменении структуры базы данных. Мощность отношения изменяется при каждом добавлении записей в файл или их удалении оттуда.

При хранении данного списка квартир в реляционной базе данных ему может соответствовать файл, в котором каждому кортежу соответствует одна запись. При распечатке или при выводе файла на экран его содержимое удобно представлять в виде таблицы. Поэтому в реляционных базах данных понятия отношения, файла и таблицы иногда отождествляются и считаются синонимами. Таким же образом иногда считаются равнозначными понятия кортежа, записи и строки.

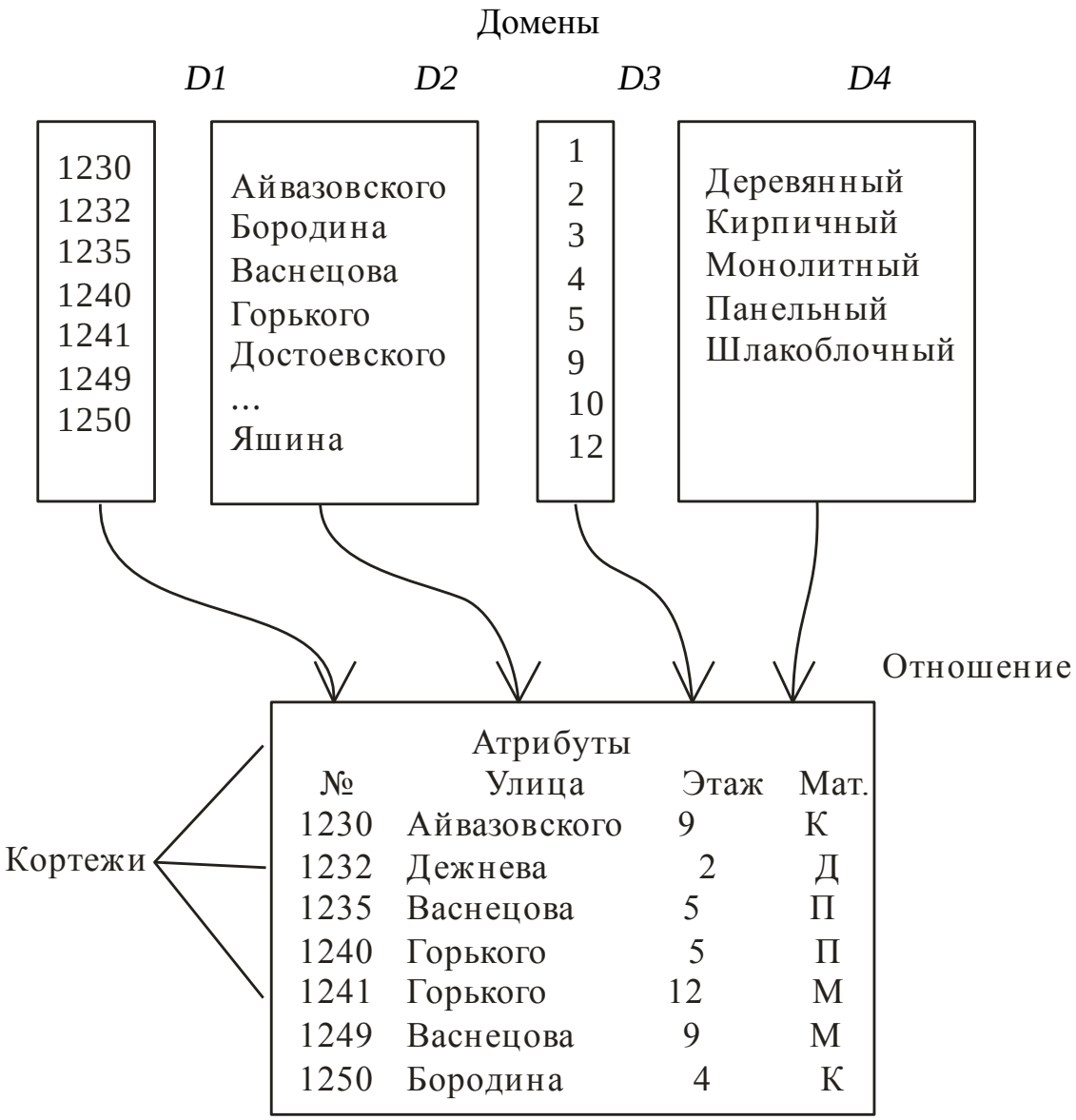


Рис. 5.11. Пример отношения

Использование ЭВМ при моделировании любой проблемной области, как правило, требует однозначности представления отображаемых объектов, явлений и процессов. Условие однозначности представления следует из формальных методов обработки данных на ЭВМ. В применении к реляционным базам данных однозначность представления требует уникальности используемых имен. В каждой базе данных должны выполняться следующие требования:

- *имена доменов* должны быть уникальными;
- *имена отношений* должны быть уникальными;
- в каждом отношении *имена атрибутов* должны быть уникальными.

Каждый *домен* представляет собой именованное множество скалярных значений одного типа. Например, домен «номер дома» – это множество всех целых чисел в диапазоне от 1 до самого большого номера дома, допустим, 2000. Домен «глубина брода» – это множество всех вещественных чисел в диапазоне

от 0,1 до 1,5. Домен «название улицы» – это список всех названий улиц (в данном городе).

Значения, образующие тот или иной домен, являются скалярными в том смысле, что являются атомарными, то есть неразложимыми, не имеющими внутренней структуры. Это означает, что скалярные значения являются *наименьшими семантическими единицами данных*. Но следует различать понятия «не имеющие структуры вообще» и «не имеющие структуры в реляционной модели». Так, например, название улицы представляет собой последовательность букв и, следовательно, имеет определенную структуру. Но в реляционной модели название улицы рассматривается как единое целое, имеющее определенный смысл.

Связь между атрибутами и доменами устанавливается при создании схемы отношения путем указания для каждого атрибута имени соответствующего домена. При этом любому атрибуту может быть поставлен в соответствие только один домен, но один и тот же домен может использоваться для создания нескольких атрибутов в разных отношениях и даже в одном отношении.

Из существования доменов следует, что сравнение значений двух атрибутов можно производить тогда, когда совпадают их домены. Если соответствующие атрибутам домены не совпадают, то сравнение атрибутов часто не имеет смысла, поскольку сравнивать можно подобные величины. Рассмотрим в качестве примера отношение, в котором имеются такие атрибуты, как рост и вес человека. Хотя оба атрибута являются вещественными числами, это величины *разного* сорта. Если рост и вес одного человека еще можно как-то сравнить, поскольку между ними существует корреляционная зависимость, то совершенно бессмысленно сравнивать вес одного человека с ростом другого. Таким образом, использование доменов налагает некоторые ограничения на сравнение и, возможно, на другие операции с атрибутами.

В сущности, домены представляют собой некоторый тип данных. Среди них различаются *встроенные типы* (целые и вещественные числа, символы) и типы данных, которые в языках программирования трактуются как *тип данных, определяемый пользователем,* или как *абстрактный тип данных.*

Когда говорят об отношениях, то различают переменные отношений и значения отношений. *Переменной отношения* называют обычную переменную, такую же, как в языках программирования, то есть именованный объект, значение которого может изменяться со временем. Значение этой переменной в тот или иной момент времени и есть *значение отношения.* Термин «отношение» обычно используется (возможно, не всегда корректно) для указания на значение отношения.

Отношение R, определенное на множестве доменов $\{D_1, ..., D_n\}$ имеет заголовок и тело. *Заголовок отношения* содержит фиксированное число пар

$$\{(A_1 : D_1), ..., (A_n : D_n)\},$$

где A_j $(j = 1, ..., n)$ – имя атрибута, а D_j $(j = 1, ..., n)$ – имя домена. Таким образом, заголовок отношения содержит описание схемы отношения. На практике заголовок отношения часто рассматривается просто как набор имен атрибутов. При этом имплицитно предполагается очевидность того, какой домен ставится в соответствие каждому атрибуту отношения.

Тело отношения содержит множество кортежей, каждый из которых содержит множество пар (имя атрибута: значение атрибута):

$\{(A_{i1} : v_{i1}), ..., (A_{in} : v_{in})\}$,

где $(i = 1, ..., m)$ и m – количество кортежей в отношении, а v_{ij} есть значение атрибута A_j из домена D_j.

Отношения первой степени называют *унарными*, второй степени – *бинарными*, третьей степени – *тернарными*, а *n*-й степени – *n-арными*.

Хотя понятия отношения и таблицы часто рассматриваются как синонимы, но если говорить точно, то отношение – это абстрактный объект,
а таблица – его графическое изображение.

В отличие от доменов, являющихся статичными, атрибуты динамичны, их значения изменяются. Когда говорят о том, что отношение *R* изменяется со временем, то имеют в виду изменение значения переменной *R*, являющейся именем отношения. Иными словами, изменяется тело отношения, а его заголовок остается постоянным. Но если изменяется заголовок отношения (состав атрибутов, а не их место в заголовке), то это будет уже другое отношение.

Как утверждается в [1, с. 100]: «Несмотря на то, что реляционная модель имеет математическую основу, отношения в реляционной модели и отношения в математике – это не одно и то же». Отношения как математические объекты обладают следующими свойствами.

1. Отношение не содержит одинаковых кортежей. Это свойство следует из определения множества: множество не содержит одинаковых элементов, а отношение является множеством кортежей. Данное свойство подчеркивает различие между отношением и таблицей: таблица может содержать одинаковые строки.

2. Кортежи отношения не являются упорядоченными. Данное свойство также следует из определения множества: любые перестановки одних и тех же элементов представляют собой одно множество.

3. Атрибуты в отношении не являются упорядоченными слева направо. Данное свойство следует из того, что заголовок (и схема) отношения определяется как *множество* атрибутов.

4. Все значения атрибутов являются атомарными. Это свойство следует из того, что все домены содержат только атомарные значения. Другая

формулировка данного свойства: отношения не содержат повторяющихся групп и значений. Отношение, отвечающее такому условию, называют *нормализованным*, или представленным в *первой нормальной форме* (1НФ). Это также означает, что в реляционной модели все отношения нормализованы. Отношение, содержащее повторяющиеся группы, называется *ненормализованным*, а процесс его преобразования в первую нормальную форму называют *нормализацией*. Причина, по которой выполняется нормализация отношений, – это простота: простота используемых элементов ведет к упрощению всей системы в целом.

При использовании реляционных систем полезно различать следующие виды отношений.

Именованное отношение – это переменная отношения, определенная в процессе описания схемы базы данных. *Производным отношением* называют отношение, полученное из других отношений с помощью тех или иных операций. *Базовым отношением* называют именованное отношение, не являющееся производным, то есть являющееся частью базы данных. *Выражаемым отношением* называется отношение, которое может быть получено из набора именованных отношений с помощью реляционного выражения. Именованное отношение всегда является выражаемым (для этого достаточно назвать имя отношения), но обратное не верно.

Именованное производное отношение называют *представлением*. *Результатом запроса* называют неименованное производное отношение, полученное в итоге выполнения некоторого запроса к базе данных, который был представлен как реляционное выражение. *Промежуточным результатом* называют неименованное производное отношение, являющееся результатом реляционного выражения, вложенного в другое реляционное выражение.

Отношение с пустым множеством кортежей, называемое *пустым отношением*, вполне корректно и является обычным отношением. Такое отношение можно сравнить с пустым файлом, то есть файлом, в котором нет ни одной записи. Отношение с пустым множеством кортежей возникает в момент, когда оно только что создано.

Каждое отношение может быть *проинтерпретировано* или *имеет интерпретацию* в терминах соответствующей предметной области. Схема (или заголовок) отношения определяет соответствующий предикат – функцию истинности. При этом атрибуты отношения играют роль аргументов функции истинности. Можно также считать, что схема отношения задает некоторую элементарную формулу исчисления предикатов. При подстановке конкретных значений аргументов (значений атрибутов) предикат превращается в конкретное высказывание, которое может быть либо истинным, либо ложным.

Любая предметная область характеризуется конечным, возможно, очень большим, числом состояний. Описание состояния предметной области в определенный момент времени может быть выполнено перечислением состояний ее объектов. В подавляющем большинстве случаев в базе данных хранятся факты, которые могут быть проинтерпретированы как истинные высказывания об объектах предметной области.

В редчайших случаях в базе данных могут храниться отношения, которые могут быть проинтерпретированы как ложные высказывания. Это осуществляется тогда, когда число ложных высказываний намного меньше числа истинных высказываний. В подобных ситуациях высказывания, отсутствующие в базе данных, считаются истинными.

Фактические значения атрибутов в некоторых кортежах некоторых отношений в тот или иной момент времени могут быть неизвестны. Поэтому множество значений каждого домена может дополняться специальным значением, которое соответствует «*неизвестному значению*», или *null-значению*. Выше об этом уже говорилось при обсуждении типов данных.

Однако из этого вполне здравого решения вытекает несколько неожиданное следствие – необходимость отказа от двузначной логики и перехода к использованию трехзначной. Чтобы убедиться в этом, рассмотрим следующий пример. Предположим, требуется сравнить значение атрибута A_k на равенство в двух кортежах i и j. Если значение одного из атрибутов при этом неизвестно, то нельзя утверждать ни то, что высказывание $v_{ik} = v_{jk}$ является истинным, ни то, что оно является ложным. Следовательно, в дополнение к двум истинностным значениям «истина» (1) и «ложь» (0) необходимо ввести значение «неизвестно» (например, -1). Если значения обоих атрибутов неизвестны, то есть им присвоено *null*-значение, то отсюда также не следует, что их значения совпадают!

Кроме того, следует различать ситуации, когда значение некоторых атрибутов неизвестно, от ситуаций, когда значение атрибута *отсутствует*. Примером отсутствующих значений атрибутов могут служить собственные названия тех или иных географических объектов. Например, в такой малонаселенной местности, как север Западно-Сибирской низменности, мелкие объекты гидрографии (незначительные реки, ручьи, озера) очень часто не имеют собственных названий. Но, с другой стороны, мы можем не знать, имеет ли конкретный объект гидрографии собственное название. При представлении в базе данных такие случаи следует различать.

Рассмотрение проблем, вызванных отсутствием значений атрибутов, нецелесообразно при первоначальном ознакомлении с реляционными моделями данных. Для их изучения следует обратиться к специальной литературе.

5.9. Реляционные базы данных

Почти все СУБД, созданные с конца 1970-х гг., основаны на реляционном подходе. Примерно с этого же времени теоретические работы в области баз данных прямо или косвенно посвящены проблемам реляционных баз данных. Некоторые авторы считают реляционную модель данных единственной существенной разработкой за всю историю существования баз данных [1].

Реляционная модель данных основана на строгих и однозначных математических принципах и по этой причине используется для изложения

теории баз данных вообще. Собственно говоря, именно для этих целей она первоначально и была разработана.

Происхождение названия «реляционная модель» связано с английским словом «relation», которое переводится как «отношение». Объяснение названия в том, что в реляционных базах данных все данные представляются пользователю только в виде таблиц. Кроме того, все результаты, получаемые с помощью предоставляемого пользователю набора операций над базой данных, также являются таблицами. С математической же точки зрения подобные таблицы являются ни чем иным, как отношениями.

Реляционные системы основаны на *реляционной модели данных*, которая представляет собой абстрактную теорию данных, построенную на таких математических теориях, как теория множеств и логика предикатов. Основные принципы реляционной модели были разработаны американским математиком Е.Ф. Коддом, впервые осознавшим, что строгие и точные математические теории могут быть использованы в области баз данных, и изложившим их в статье [9]. Как писал К.Дж. Дейт «С этой статьи все и началось. И хотя прошло более четверти века, она остается актуальной и стоит того, чтобы ее перечитывали. Конечно, некоторые идеи были несколько улучшены со времени публикации этой статьи, но по своей природе это были эволюционные, а не революционные изменения. Кроме того, в статье есть некоторые идеи, следствия которых до сих пор полностью не исследованы» [1, с. 97].

Реляционные системы появились как ответ на возрастание сложности создаваемых систем обработки данных, когда повышение эффективности их использования могло идти двумя путями:

- либо представлением специалистам по информационным технологиям новых эффективных инструментальных средств;
- либо представлением пользователям возможности непосредственного доступа к базам данных, полностью игнорируя специалистов по информационным технологиям.

Последняя альтернатива возможна только в том случае, если объекты данных будут достаточно просты и их поведение для пользователя также будет простым и предсказуемым. Такими объектами являются таблицы.

Среди других преимуществ реляционного подхода особого подчеркивания заслуживает наличие *солидной математической базы*.

Формальная теория, на основании которой строятся реляционные системы, называется *реляционной моделью*. Объектами реляционной модели являются отношения (таблицы), и они изучаются только на логическом уровне, вопросы физической реализации не рассматриваются. В реляционной модели рассматриваются структуры данных, целостность данных, обеспечиваемая первичными и внешними ключами, и операции над данными.

Реляционная база данных является такой базой данных, в которой вся хранимая в ней информация представлена в виде совокупности отношений. Пример базы данных, содержащей несколько отношений, приводится на рис. 5.12. Данная база данных содержит три разновидности данных: о продаваемых квартирах, о домах, в которых находятся продаваемые квартиры,

и о владельцах квартир. Кроме того, в отношении КВАРТИРА содержатся сведения о связях между квартирами и домами, а также между продаваемыми квартирами и продавцами. Подобная база данных могла быть создана в агентстве недвижимости.

ДОМ

Дом	Улица	№ дома	Ч. этажей	Мат.
1	Айвазовского	1	5	П
112	Айвазовского	24	5	К
113	Бородина	1	2	Д
245	Васнецова	8	12	М

КВАРТИРА

Кв.	Дом	№ кв.	Ч. комн.	Площ.	Влад.
4	112	5	2	42,5	12
2	245	21	1	28,3	77
1	113	36	3	59,2	33
3	1	5	2	44,5	77

ВЛАДЕЛЕЦ

Влад.	Фамилия	Имя	Отч.	Тлф
77	Петров	Петр	Петрович	1231231
12	Сидоров	Иван	Петрович	2223334
33	Иванов	Иван	Иванович	3322233

Рис. 5.12. Пример базы данных агентства недвижимости

Пример другой базы данных приводится на рис. 5.13. Эта база данных, которую назовем «Обеды», используется гипотетическим предприятием общественного питания, обслуживающим соседние организации. При этом предполагается, что служащие каждой обслуживаемой организации могут через Internet ознакомиться с меню на завтрашний день и заказать необходимые блюда.

МЕНЮ

#Блюда	Название	Цена	Ч. порц.
1	Ассорти мясное	40,9	50
25	Борщ	35,9	70
112	Винегрет	19,2	50
…			
18	Шашлык	80,0	30
13	Щи	25,5	20

ОРГАНИЗАЦИИ

#Орг	Название	Улица	Дом	Офис	Тлф
4	ЗАО «Ракурс»	Мясницкая	406	606	9282001
2	ООО «Восток»	Васнецова	115	130	1112223
1	АБ «Акцепт»	Бородина	77	6	3211233
23	ТД «Эльдорадо»	Горького	18	204	7788445

ЗАКАЗЫ

#Орг	#Блюда	З. порц
23	18	20
23	112	5
23	1	15
2	112	35
2	13	35
…		

Рис. 5.13. Пример базы данных «Обеды»

База данных «Обеды» состоит из трех отношений. Отношение «Меню» содержит такие сведения о блюдах, как его уникальный идентификатор (номер) #блюда, название, цену и число изготовленных порций. С этой информацией потенциальные клиенты могут ознакомиться через Internet. По мере поступления заказов на каждое блюдо уменьшается соответствующее число порций.

В отношении «Организация» содержится информация об обслуживаемых организациях: уникальный номер #орг, присвоенный каждой организации, ее название, адрес (улица, номер дома и номер офиса в здании) и телефон. Название организации не может служить ее идентификатором, поскольку не исключено существование двух или более организаций с одинаковыми названиями.

Отношение «Заказы» содержит атрибуты «#орг», «#блюда» и заказанное «число порций».

В реляционных базах данных каждое отношение хранится в отдельном файле. Структура каждого файла определяется в процессе проектирования базы данных. Все записи одного файла имеют одинаковый формат и одинаковую длину. Поэтому по начальному адресу файла на носителе данных и номеру логической записи в файле легко получить доступ к ее содержанию.

Для идентификации кортежей в отношениях используются первичные ключи. *Первичным ключом* отношения называют атрибут или набор атрибутов, который может быть использован для однозначной идентификации конкретного кортежа. Важным свойством первичного ключа является его *неизбыточность*. Это означает, что при исключении хотя бы одного атрибута из первичного

ключа с помощью оставшихся атрибутов нельзя однозначно идентифицировать кортеж.

Первичный ключ представляет собой *идентификатор* отношения *R*. Но первичный ключ является частным случаем более общего понятия – потенциального ключа. *Потенциальным* (или *возможным*) *ключом* называют подмножество атрибутов отношения *R*, обладающее свойствами:

– *уникальности*, означающей, что в *R* нет двух кортежей с одинаковыми значениями потенциальных ключей;

– *неизбыточности*, в соответствии с которой никакое подмножество потенциального ключа не является потенциальным ключом.

Потенциальный ключ, состоящий из нескольких атрибутов, называют *составным*, а ключ, состоящий из одного атрибута, – *простым ключом*. Каждое отношение имеет хотя бы один потенциальный ключ. Это следует из того, что отношение не содержит повторяющихся кортежей. Поэтому, в крайнем случае, потенциальным ключом является все множество атрибутов отношения *R*.

Значение потенциальных ключей состоит в том, что они обеспечивают *основной механизм адресации на уровне кортежей.* Реляционные системы гарантируют единственный способ точного указания любого кортежа в отношении – значение некоторого потенциального ключа. Если сравнивать реляционные системы с ЭВМ, то потенциальные ключи имеют такое же фундаментальное значение для работы реляционной системы, как адресация оперативной памяти для ЭВМ.

В реляционной модели один из потенциальных ключей выбирается в качестве *первичного ключа*, а остальные потенциальные ключи называют *альтернативными ключами*.

Организация различных отношений в базу данных как единое целое осуществляется с помощью внешних ключей. Если имеются отношение *R1* с потенциальным ключом *K1* и отношение *R2* с подмножеством атрибутов *K2*, и с каждым значением *K2* совпадает некоторое значение потенциального ключа *K1*, то подмножество *K2* называется *внешним ключом*.

В определении внешнего ключа важно отметить следующие моменты:

– внешние ключи, как и потенциальные, определяются как множества атрибутов;

– каждое значение внешнего ключа должно совпадать с одним из значений соответствующего потенциального ключа;

– внешний ключ может быть простым или составным;

– каждый атрибут, входящий в состав внешнего ключа, должен быть определен на том же домене, что и соответствующий атрибут соответствующего потенциального ключа;

– для внешнего ключа не требуется, чтобы он являлся компонентом первичного ключа в содержащем его отношении.

Отношение, которое содержит внешний ключ, называется *ссылающимся отношением*, а отношение, содержащее соответствующий первичный ключ, называют *ссылочным отношением*. Таким образом, в нашем определении

внешнего ключа отношение *R1* является ссылочным, а *R2* – ссылающимся отношением. Значение внешнего ключа представляется *ссылкой* к кортежу, содержащему соответствующее значение потенциального ключа, такой кортеж называют *ссылочным,* или *целевым*. Поэтому для целостности реляционной базы данных крайне важно отсутствие неверных значений внешних ключей, то есть таких значений, для которых нет соответствующих значений потенциальных ключей. Данная проблема в теории реляционных моделей получила название проблемы *ссылочной целостности*.

В отношении «Организация» первичным ключом является атрибут #орг, а в отношении «Заказ» функцию первичного ключа выполняет пара атрибутов (#орг, #блюда). Нетрудно видеть, что в последнем отношении ни один кортеж не может быть идентифицирован с помощью только одного из атрибутов, составляющих первичный ключ. Атрибут #орг в отношении «Заказ» является внешним ключом для отношения «Организация».

В соответствии с теорией реляционных БД, в отношении не может быть двух и более полностью идентичных кортежей (строк, записей). Более того, в отношении не может быть двух кортежей с одинаковыми значениями первичных ключей. Наличие в отношении двух кортежей с идентичными первичными ключами порождает проблемы с интерпретацией имеющихся данных. Допустим, в отношении «Заказы» среди прочих существуют два кортежа.

23	12	10
…	…	…
23	12	11

В подобных ситуациях непонятно, каким образом трактовать такие кортежи. Возможно, что правильным является первый кортеж, а второй ошибочен, но не исключено, что правилен второй кортеж. Кроме того, возможно, что правильны оба кортежа и заказчик 23 заказал 21 блюдо под номером 12.

В полнофункциональных реляционных СУБД попытки создать кортеж с ключом, совпадающим с ключом другого кортежа, уже имеющегося в отношении, обнаруживаются и пресекаются системой автоматически. Но некоторые СУБД допускают существование кортежей с одинаковыми значениями первичных ключей. В таких случаях контроль корректности отношений возлагается на пользователя.

Занесение данных в базу данных часто носит произвольный характер; обычно данные заносятся в БД по мере их появления. Так, в нашем последнем примере порядок обращения клиентов к базе данных никак не фиксирован, и они могут делать заказы в любое удобное для них время. Клиенты также могут заказывать блюда в любой последовательности. Кроме того, они могут обращаться к базе данных несколько раз, делая новые заказы, а также аннулируя или исправляя ранее сделанные. В итоге записи в файле «Заказы» будут расположены некоторым случайным образом. Как правило, в реальных базах данных записи в файлах данных также никак не упорядочены.

В небольших по объему базах данных поиск нужного кортежа в том или ином отношении не является проблемой. Отношение можно последовательно просмотреть за приемлемое время и либо найти нужную запись, либо убедиться в ее отсутствии. Однако в базах данных, насчитывающих миллионы записей и более, такая стратегия может оказаться катастрофически неэффективной. Значительное ускорение поиска нужных кортежей в отношении обеспечивается с помощью индексных файлов.

Индексом называют атрибут или набор атрибутов, предназначенный для организации записей в том или ином порядке и ускорения поиска необходимых записей в файле данных. В качестве индекса чаще всего используется первичный ключ. Для создания индексов могут также использоваться атрибуты, не входящие в первичный ключ; такие индексы называют *вторичными*.

Рассмотрим создание индексного файла на примере отношения «Организации», изображенного на рис. 5.14. В действительности такой файл данных может содержать множество других атрибутов, например, фамилию, имя, отчество и телефон первого руководителя, фамилию, имя, отчество и телефон контактного лица, номер договора на обслуживание, сроки действия договора, сумму внесенного аванса, расчетный счет и т. п. Поскольку пример носит учебный характер, на рис. 5.14 эти данные опущены.

Файл данных «Организации»

№ записи	#орг	Название	Улица	Дом	Офис
1	4	ЗАО «Ракурс»	Мясницкая	40/6	605
2	2	ООО «Восток»	Васнецова	115	130
3	1	АБ «Акцепт»	Бородина	77	6
4		**Удаленная запись**			
5	23	ТД «Эльдорадо»	Горького	18	24

Индексный файл «Организации»

№ записи в индекс. файле	#орг	№ записи в файле данных
1	1	3
2	2	2
3	4	1
4	23	5

Рис. 5.14. Файл данных и индексный файл

Для нас в данном случае важно то, что, во-первых, каждая запись может быть достаточно длинной, и, во-вторых, эти записи никак не упорядочены. Кроме того, файл содержит и удаленные записи. (В реляционных БД отдельные кортежи могут удаляться не физически, а логически. Это означает, что соответствующая запись не удаляется из файла, для чего потребовалось бы смещение всех последующих записей на одну строку вверх, а в специальном служебном поле записи делается отметка об ее удалении, и в дальнейшем при чтении эта запись игнорируется.)

Индексный файл содержит последовательность записей, упорядоченную по значению первичного ключа. Пример индексного файла приведен на рис. 5.14. В действительности для создания индексных файлов используются древовидные структуры. Длина каждой записи в индексном файле обычно намного короче, чем длина записей в соответствующем файле данных. Поэтому доступ к нужной записи файла данных через индексный файл может осуществляться очень быстро.

Процесс создания индексных файлов называют *индексированием.* В некоторых СУБД индексирование файлов данных осуществляется автоматически, в других – принудительно по команде пользователя.

Создание базы данных начинается с создания концептуальной модели предметной области. С этой целью составляется перечень всех сущностей (объектов, процессов, …), их свойств и связей между сущностями (рис. 5.15).

Название	Название атрибута	Тип
Идентификатор блюда	#блюда	Целое (3)
Название блюда	Назв_блюда	Симв. (32)
Цена	Цена	Веществ. (6.2)
Изготовленное число порций	Впорц	Целое (3)
Идентификатор организации	#орг	Целое (3)
Название организации	Назв_орг	Симв. (40)
Улица	Ул	Симв. (32)
Дом	Дом	Симв. (5)
Офис	Оф	Целое (3)
Телефон организации	Тлф	Целое (7)
Заказанное число порций	Чпорц	Целое (2)

Отношения:

1) Меню (#блюда, назв_блюда, впорц, цена);
2) Клиенты (#орг, назв_орг, ул, дом, оф, тлф);
3) Заказы (#орг , #блюда, чпорц).

Рис. 5.15. Пример концептуальной модели

В данном примере в графе «тип» указан тип элементов данных и их длина, для вещественных чисел указывается число знаков после десятичной точки.

Описание концептуальной модели и схемы базы данных осуществляется на языке описания данных используемой СУБД. Продумывание концептуальной модели может потребовать много времени, но ее описание и создание схемы БД на языке описания данных осуществляется быстро.

После ввода описания схемы базы данных СУБД создает незаполненную (пустую) базу данных. Ввод данных осуществляется последовательно запись за записью с помощью команд записи. В процессе ведения базы данных записи могут добавляться, корректироваться и удаляться с использованием набора команд. Таким образом, состояние БД постоянно изменяется, отображая действительное состояние предметной области. Состояние того или иного отношения в некоторый момент времени называют *экземпляром отношения*.

5.10. Цели проектирования

Проблема *проектирования реляционной базы данных* сводится к поиску ответов на вопросы: каков должен быть перечень отношений и какова должна быть структура каждого отношения? При этом имеется в виду логическое проектирование. Проблемы физического проектирования, как правило, не рассматриваются не потому, что их значение для будущей БД незначительно, наоборот, оно очень велико, а потому что эти вопросы довольно специфические и существенным образом зависят от используемой СУБД. Проектирование логической структуры совершенно не зависит от применяемой СУБД и основывается на некотором числе общих и строгих теоретических принципов.

Основными целями при проектировании реляционных баз данных являются:

1) обеспечение возможности хранения всех необходимых данных в одной базе данных;
2) исключение избыточности данных;
3) минимизация числа отношений, хранимых в БД;
4) нормализация отношений для решения некоторых проблем, связанных с ведением БД.

Для решения первой указанной проблемы проектировщик базы данных изучает проблемную область и информационные потребности всех потенциальных пользователей базы данных. Итогом изучения предметной области должен быть полный перечень атрибутов, представляющих интерес. После этого проектировщик должен решить вопрос о структуре БД, определить все отношения и схему каждого отношения.

Задача исключения избыточности данных не является сама собой разумеющейся. Для ее понимания следует различать *дублирование данных* и *избыточное дублирование данных*. Рассмотрим отношение, содержащее табельные номера служащих и номера отделов, в которых они работают (рис. 5.16).

Служащий	Отдел
123	7
124	26
127	26
131	23
132	7

Рис. 5.16. Пример дублирования данных

В этом отношении можно обнаружить дублирование одних и тех же данных: номер каждого отдела будет появляться столько раз, сколько в нем сотрудников. Но это повторение данных не является избыточным. Если у сотрудников 127 и 132 удалить номер отдела, то невозможно будет определить номер их отдела.

Дополним данное отношение новым атрибутом – фамилией, именем и отчеством начальника соответствующего отдела.

Полученное отношение характеризуется наличием избыточности, поскольку в нем данные о начальниках отделов дублируются многократно (рис. 5.17). Чтобы избавиться от указанного недостатка, повторяющиеся данные заменим пробелами и получим следующее отношение (рис. 5.18).

Служащий	Отдел	Начальник отдела
123	7	Борисов Сергей Иванович
124	26	Васильев Николай Николаевич
127	26	Васильев Николай Николаевич
131	23	Андреев Владимир Иосифович
132	7	Борисов Сергей Иванович

Рис. 5.17. Пример избыточного дублирования

Данное отношение содержит всю необходимую информацию, но подобных решений следует избегать. Причина в том, что отношения не должны содержать пустых полей, так как при их наличии требуются дополнительные затраты на программирование. Кроме того, использованный нами способ борьбы с избыточностью потенциально опасен. Если, например, сотрудник 123 уволится, то соответствующая запись будет удалена из базы данных, а вместе с ней из БД исчезнет информация о начальнике 7-го отдела.

Служащий	Отдел	Начальник отдела
123	7	Борисов Сергей Иванович
124	26	Васильев Николай Николаевич
127	26	
131	23	Андреев Владимир Иосифович
132	7	

Рис. 5.18. Пример исключения избыточности

В данном случае более правильным методом борьбы с избыточностью данных является замена одного отношения двумя показанными ниже отношениями (рис. 5.19).

Разбиение одного отношения на несколько отношений является типовым методом проектирования структуры реляционных баз данных. Но возрастание числа отношений в базе данных влечет за собой определенные неудобства для пользователей, в связи с чем и появляется третья цель при проектировании баз данных – минимизация числа отношений в БД. Поэтому задача проектировщика заключается в том, чтобы найти разумный компромисс между противоречивыми требованиями исключения избыточности данных и минимизации числа отношений.

Служащий	Отдел
123	7
124	26
127	26
131	23
132	7

Отдел	Начальник отдела
7	Борисов Сергей Иванович
23	Андреев Владимир Иосифович
26	Васильев Николай Николаевич

Рис. 5.19. Пример корректного устранения избыточности

Выше было показано, что при неудачном определении структуры данных существует опасность потери данных. В других случаях аналогичным образом возникает опасность искажения данных при обновлении. Задачей проектировщика является выявление таких потенциально опасных отношений и их *нормализация*, под которой понимают определенный способ оптимального разбиения отношений.

5.11. Универсальное отношение

После определения концептуальной модели предметной области следует *разработка схемы БД*, под которой понимают определение числа отношений и структуры каждого отношения. В небольших по объему базах данных разработка их структуры может начинаться с определения универсального отношения. *Универсальным отношением* называют отношение, содержащее все необходимые атрибуты и удовлетворяющее информационные потребности всех пользователей БД.

Рассмотрим универсальное отношение на следующем примере. Пусть в некотором городе городская администрация приняла решение о контроле над сдачей в аренду производственных площадей муниципальными предприятиями, и с этой целью требуется создать базу данных.

Предположим далее, что в результате изучения информационных потребностей была установлена необходимость представления следующих атрибутов.

1. Здание – уникальное символьное значение, обозначающее улицу и номер дома.
2. МУП (название муниципального унитарного предприятия) – уникальное символьное значение, поскольку в одном городе не может быть двух муниципальных предприятий с одинаковыми наименованиями. Одному предприятию может принадлежать несколько зданий.
3. Тлф (телефон руководителя предприятия) – уникальное целое значение. У руководителя может быть лишь один городской телефон, с другой стороны, каждый телефон может принадлежать только одному руководителю предприятия.
4. Арендатор (название фирмы, арендующей производственные площади) – символьное значение. Не исключается, что могут существовать фирмы с одинаковыми наименованиями.
5. Месяц – представляется в формате «*годмесяц*», где год и месяц – целые числа. Например, январь 2005 г. представляется как 200501.
6. Площадь (S) – размер общей площади, арендуемой конкретным арендатором в одном здании. Один арендатор может арендовать площади в разных зданиях, в том числе – у разных муниципальных предприятий. В разные

месяцы арендатор может арендовать различную площадь в одном и том же здании.

7. Плата (размер арендной платы за 1 кв. м) – вещественное значение. Считается, что в одном здании арендная плата за 1 кв. м является одной и той же для всех арендаторов, но может изменяться с течением времени.

Данный пример носит учебный характер, поэтому атрибуты, которые были бы необходимы в реальной геоинформационной системе (ФИО руководителя МУП, ИНН каждого юридического лица, геопространственные данные о здании и т. п.), здесь опущены как несущественные для понимания универсального отношения.

Будем считать, что в некоторый момент времени необходимые данные об аренде можно представить в виде, изображенном на рис. 5.20.

Здание	МУП	Тлф	Арендатор	Месяц	S	Плата
Бабеля 125	Дом быта	123456	Норд	200501	50	1 000
			Вест	200502	100	1 000
				200503	150	1 200
Гоголя 2	Автокомбинат 1	234567	Норд	200503	60	500
Бабеля 1	Автокомбинат 2	345678	АББ	200501	75	400
				200502	75	400
				200503	75	500
Гаршина 1	Горзеленхоз	121212	Аг. «Travel»	200501	20	600
				200502	20	600
Чехова 22	Горпроект	123123	АБ «Алемар»	200502	250	1 000

Рис. 5.20. Представление данных об аренде

Хотя показанная на рис. 5.20 таблица могла бы храниться в базе данных, она *не может считаться отношением* по следующей причине. Атрибуты «Здание», «МУП» и «Тлф» являются атомарными, а остальные же – множественными. Таким образом, элементы не всех кортежей представляют собой атомарные значения. Чтобы избавиться от указанного недостатка таблицы, ее необходимо преобразовать в отношение вставкой недостающих элементов для каждого множественного значения. В результате такого преобразования получим таблицу на рис. 5.21.

Здание	МУП	Тлф	Арендатор	Месяц	S	Плата
Бабеля 125	Дом быта	123456	Норд	200501	50	1000
Бабеля 125	Дом быта	123456	Вест	200502	100	1000
Бабеля 125	Дом быта	123456	Вест	200503	150	1200
Гоголя 2	Автокомбинат 1	234567	Норд	200503	60	500
Бабеля 1	Автокомбинат 2	345678	АББ	200501	75	500
Бабеля 1	Автокомбинат 2	345678	АББ	200502	75	400
Бабеля 1	Автокомбинат 2	345678	АББ	200503	75	400
Гаршина 1	Горзеленхоз	121212	Аг. «Travel»	200501	20	600
Гаршина 1	Горзеленхоз	121212	Аг. «Travel»	200502	20	600
Чехова 22	Горпроект	123123	АБ «Алемар»	200502	250	1 000

Рис. 5.21. Пример корректного универсального отношения

Данная таблица является вполне корректным экземпляром отношения, но характеризуется большой избыточностью, что также является недостатком и

причиной потенциальных проблем в процессе ведения базы данных. Различаются три вида таких проблем, называемых также *аномалиями*: добавления, обновления и удаления кортежей.

Проблема добавления кортежей возникнет тогда, когда появится новое здание и сведения о нем потребуется занести в БД. Но если отсутствует хотя бы один арендатор площадей в этом здании, то данные о здании не могут быть помещены в БД, так как в ней не должны присутствовать пустые (или нулевые) значения. Если проигнорировать это требование и разместить в БД данные о новом здании, то получим таблицу, представленную на рис. 5.22.

Здание	МУП	Тлф	Арендатор	Месяц	S	Плата
Бабеля 125	Дом быта	123456	Норд	200501	50	1 000
Бабеля 125	Дом быта	123456	Вест	200502	100	1 000
Бабеля 125	Дом быта	123456	Вест	200503	150	1 200
Гоголя 2	Автокомбинат 1	234567	Норд	200503	60	500
Бабеля 1	Автокомбинат 2	345678	АББ	200501	75	500
Бабеля 1	Автокомбинат 2	345678	АББ	200502	75	400
Бабеля 1	Автокомбинат 2	345678	АББ	200503	75	400
Гаршина 1	Горзеленхоз	121212	Аг. «Travel»	200501	20	600
Гаршина 1	Горзеленхоз	121212	Аг. «Travel»	200502	20	600
Чехова 22	Горпроект	123123	АБ «Алемар»	200501	250	1 000
Новая 1	Бюро тех. инв.	333333			0	0

Рис. 5.22. Пример аномалии добавления

Но подобное решение не является удовлетворительным, так как в некоторых случаях сведения не будут соответствовать действительности. Такое искажение может произойти, например, при работе программы, вычисляющей среднее значение платы за 1 кв. м арендуемой площади.

Проблема обновления при большом числе избыточных данных возникает практически всегда. Ее суть заключается в нарушении целостности данных, появлении в БД противоречивых сведений. Так, например, если в некотором МУП изменился номер телефона руководителя, то необходимо отыскать в отношении все кортежи со зданиями, принадлежащими этому МУП, и изменить в них значение номера телефона для всех арендаторов в соответствующих зданиях.

Проблема удаления возникает при удалении кортежей, содержащих уникальные сведения. Допустим, МУП «Автокомбинат 1» потерял своего единственного арендатора. Тогда соответствующий кортеж необходимо удалить из универсального отношения, после чего в базе данных исчезнут все данные об этом МУП.

Приведенные выше нарушения целостности БД, возникающие при ее ведении, называют соответственно *аномалиями добавления, изменения и удаления*.

Таким образом, использование универсального отношения может быть связано с определенными проблемами, которые проектировщик БД должен предвидеть. Его задача – разработать схему базы данных таким образом, чтобы

по возможности исключить подобные потенциальные проблемы, используя нормализацию отношений.

5.12. Использование функциональных зависимостей

Один из способов нормализации отношений основан на анализе функциональных зависимостей между атрибутами. Но предварительно каждое отношение должно быть представлено в первой нормальной форме. По определению отношение находится в *первой нормальной форме* (1НФ) тогда, когда каждый атрибут имеет и всегда будет иметь атомарное (скалярное) значение. Примером отношения в 1НФ является отношение, представленное на рис. 5.21. Процедуру разбиения отношения на несколько отношений с целью уменьшения вероятности появления аномалий называют *декомпозицией.*

Функциональную зависимость между атрибутами отношения определяют следующим образом. Пусть даны два атрибута *X* и *Y*. Говорят, что *Y функционально зависит* от *X*, если каждому значению *X* соответствует ровно одно значение *Y*. Функциональная зависимость между атрибутами означает, что во всех кортежах, имеющих значение x_i атрибута *X*, будет одно и то же значение y_j атрибута *Y*. Оба атрибута при этом могут быть не только атомарными, но и составными (группами).

При использовании математического способа записи факта функциональной зависимости между атрибутами *X* и *Y* используется стрелка: $X \to Y$. При графическом способе записи связей между атрибутами также используется стрелка (рис. 5.23).

Функциональная зависимость между атрибутами может быть установлена только в результате изучения связей в предметной области. Так, в примере с базой данных о сдаче в аренду площадей функциональной является зависимость между зданием и его владельцем (МУП). Другой пример: у предприятия может быть только один первый руководитель (принцип единоначалия).

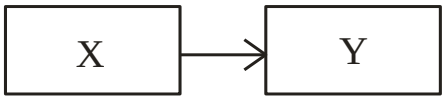


Рис. 5.23. Функциональная зависимость

На рис. 5.24 показано графическое представление функциональной зависимости между атрибутами. Здесь следует обратить внимание на тот факт, что зависимость между атрибутами МУП и Тлф является взаимной: на каждом предприятии существует только один телефон руководителя и каждое предприятие однозначным образом определяется телефоном руководителя. Данное обстоятельство (взаимная функциональная зависимость) на рис. 5.24 отображено с помощью двунаправленной стрелки.

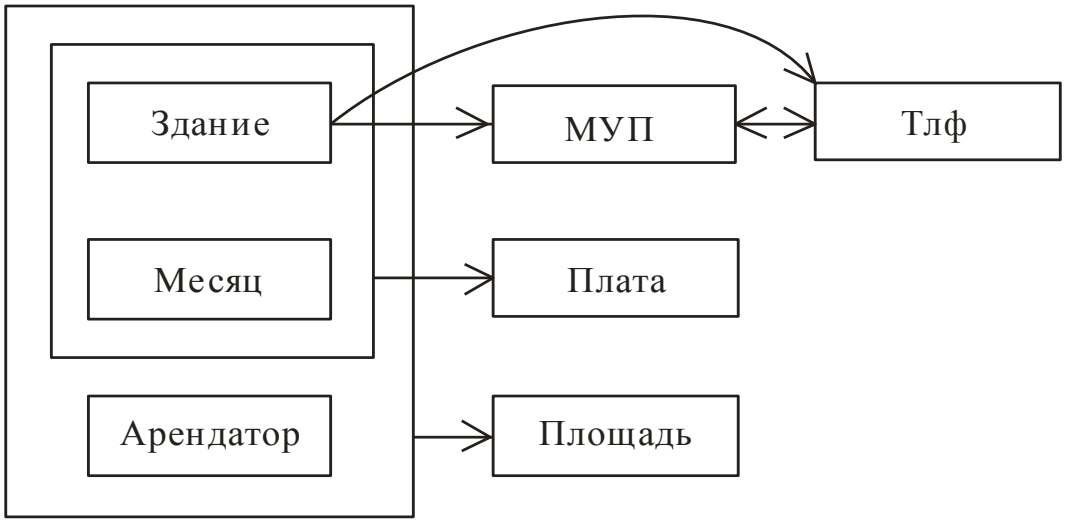


Рис. 5.24. Графическое представление зависимостей

При анализе зависимостей между атрибутами с целью избавления от нежелательных свойств отношений используется понятие возможного ключа. Напомним, что *возможным ключом* называют атрибут или набор атрибутов, который может быть использован в качестве первичного ключа рассматриваемого отношения. Таким образом, первичный ключ всегда является возможным ключом. Но если первичный ключ для некоторого отношения уже выбран, то не исключено существование другого атрибута или совокупности атрибутов, которые также могли бы однозначно идентифицировать каждый кортеж в отношении.

Если существует функциональная зависимость $X \to Y$ и Y функционально не зависит от любого подмножества X, то X называют *детерминантом* Y.

В отношении на рис. 5.24 существует только один возможный ключ – тройка атрибутов *(Здание, Месяц, Арендатор)*. Этот набор атрибутов однозначным образом определяет значения всех других атрибутов в отношении:

– атрибуту *Здание* соответствует только одно значение атрибутов *МУП* и *Тлф*, то есть, имеют место зависимости

Здание $\to$ МУП,

Здание $\to$ Тлф;

– значение атрибута *Плата* однозначно определяется значением пары атрибутов *(Здание, Месяц)* или

(Здание, Месяц) $\to$ Плата;

– значение атрибута *Площадь* однозначным образом зависит от триады *(Здание, Месяц, Арендатор)* или

(Здание, Месяц, Арендатор) $\to$ Площадь.

Кроме того, существует отмеченная выше обратная функциональная зависимость

Тлф $\to$ МУП.

Детерминантами в данном универсальном отношении являются левые части всех функциональных зависимостей: *(Здание)*, *(Здание, Месяц)*, *(Здание, Месяц, Арендатор)*. Каждая взаимная функциональная зависимость порождает

два детерминанта, поэтому *(Тлф)* также является детерминантом. Таким образом, всего в обсуждаемом отношении имеется четыре детерминанта.

Коддом было получено формальное доказательство важного свойства отношений: большинство потенциальных аномалий будет устранено, если и только если каждый детерминант отношения является возможным ключом. Об отношении, отвечающем данному условию, говорят, что оно находится в *нормальной форме Бойса – Кодда* (НФБК).

Известны другие, в том числе более сильные ограничения на свойства разрабатываемых отношений, называемые *нормальными формами*. Такими формами являются уже упоминавшаяся 1-я, а также 2, 3 и 4-я нормальные формы. Однако, при разработке схем баз данных часто ограничиваются приведением отношений к НФБК.

Чтобы проверить принадлежность нашего отношения к НФБК, перечислим все детерминанты и все возможные ключи и проверим, является ли каждый детерминант возможным ключом.

На рис. 5.25 легко видеть, что не каждый детерминант является возможным ключом. Поэтому нашей ближайшей задачей является такая декомпозиция универсального отношения, в результате которой полученные отношения будут находиться в нормальной форме Бойса – Кодда.

Возможные ключи	Детерминанты
(Здание, Месяц, Арендатор)	*(Здание, Месяц, Арендатор)*
	(Здание, Месяц)
	(Здание)
	(МУП)
	(Тлф)

Рис. 5.25. Детерминанты и ключи

5.13. Общий алгоритм декомпозиции отношений

Алгоритм декомпозиции отношений с целью их приведения к НФБК разбивается на четыре этапа:

1) представление базы данных в виде универсального отношения;
2) определение всех функциональных зависимостей между атрибутами;
3) проверка, находится ли каждое отношение в НФБК. Если для некоторого отношения это условие выполняется, то процесс его декомпозиции заканчивается; в противном случае отношение разбивается на два отношения;
4) повторение шагов 2 и 3 для каждого полученного в результате декомпозиции отношения до тех пор, пока все отношения не будут приведены к НФБК.

Декомпозиция отношения, не находящегося в нормальной форме Бойса – Кодда, осуществляется следующим образом. Пусть имеется отношение $R(X_1, ..., X_k)$, не приведенное к НФБК. Допустим, что в результате анализа в нем обнаружена функциональная зависимость $X_i \to X_j$, являющаяся причиной того, что R не находится в НФБК. Это будет в том случае, когда X_i является детерминантом, но не является ключом. Тогда создаются два новых

отношения $R_1(X_1, ..., X_i, ..., X_{j-1}, X_{j+1}, ..., X_k)$ и $R_2(X_i, X_j)$, где R_1 получено из R путем исключения функциональной зависимости, а R_2 получено на основе зависимости $X_i \to X_j$ (рис. 5.26). Говорят, что отношение R_2 является *проекцией* отношения R, а подобный способ разбиения отношений называют *декомпозицией без потерь* при естественном соединении.

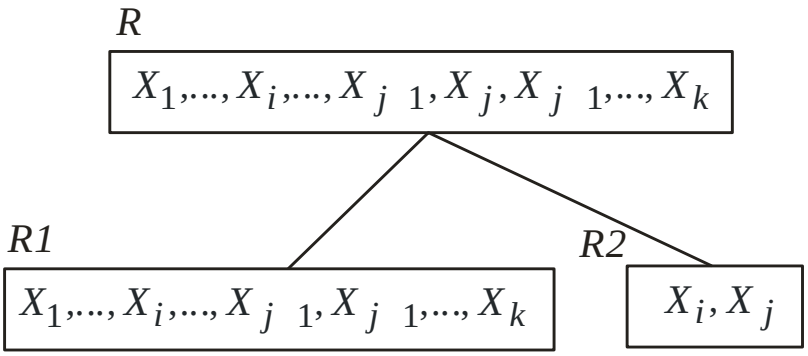


Рис. 5.26. Разбиение отношения

Применим указанный способ декомпозиции к отношению *Аренда,* которое формально можно записать как

Аренда (Здание, МУП, Тлф, Арендатор, Месяц, Площадь, Плата).

Из рис. 5.25 следует, что не являются ключами четыре детерминанта *(Здание, Месяц)*, *(Здание)*, *(МУП)* и *(Тлф)*. В выборе функциональной зависимости, используемой для разбиения отношений, всегда имеется некоторая свобода, но при этом не исключено, что различные пути могут привести к разным базам данных. Вообще же для выбора функциональной зависимости рекомендуется поиск цепочки зависимостей вида $X \to ...Y \to Z$ и использование правой крайней зависимости.

В нашем универсальном отношении *Аренда* такой цепочкой является последовательность *Здание* $\to$ *МУП* $\to$ *Тлф*. Следовательно, кандидатом на первую проекцию служит зависимость *МУП* $\to$ *Тлф*. Исключив данную зависимость из отношения *Аренда*, получим два отношения

Аренда2 (Здание, МУП, Арендатор, Месяц, Площадь, Плата);

Аренда3 (МУП, Тлф).

Эти же отношения представлены на рис. 5.27.

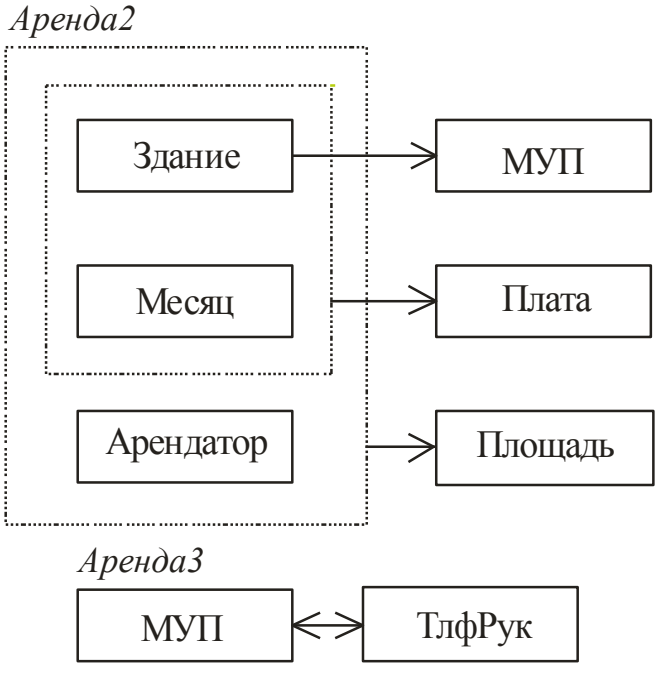


Рис. 5.27. Разбиение отношения *Аренда*

В отношении *Аренда3* оба детерминанта *(МУП)* и *(Тлф)* являются возможными ключами. Следовательно, отношение *Аренда3* находится в НФБК и в дальнейшей декомпозиции не нуждается.

В отношении *Аренда2* возможный ключ только один *(Здание, Арендатор, Месяц,)*, но три детерминанта функциональных зависимостей: *(Здание, Арендатор, Месяц), (Здание, Месяц), (МУП)*. Поэтому оно должно быть подвергнуто разбиению. Для образования новых отношений используем проекцию отношения *Аренда2*, порождаемую функциональной зависимостью *Здание* $\rightarrow$ *МУП*. Тогда получим два новых отношения

Аренда4 (Здание, Арендатор, Месяц, Площадь, Плата),

Аренда5 (Здание, МУП).

Отношение *Аренда5* содержит функциональную зависимость, детерминант которой *Здание* является возможным ключом. Таким образом, указанное отношение приведено к окончательному виду, то есть к нормальной форме Бойса – Кодда.

В *отношении Аренда4* содержится две функциональных зависимости:

(Здание, Арендатор, Месяц) $\rightarrow$ *(Площадь)*;

(Здание, Месяц) $\rightarrow$ *(Плата)*.

Детерминант первой функциональной зависимости является возможным ключом, но детерминант второй зависимости возможным ключом не является. Отсюда следует, что эта функциональная зависимость должна быть использована для декомпозиции отношения *Аренда4*. В итоге мы получим два новых отношения

Аренда6 (Здание, Арендатор, Месяц, Площадь),

Аренда7 (Здание, Месяц, Плата).

Оба полученных отношения находятся в НФБК, следовательно, процесс декомпозиции универсального отношения можно считать законченным.

Таким образом, база данных *Аренда* представляет собой совокупность отношений

Аренда3 (МУП, Тлф),

Аренда5 (Здание, МУП),

Аренда6 (Здание, Арендатор, Месяц, Площадь),

Аренда7 (Здание, Месяц, Плата).

В *графическом* виде эти отношения изображены на рис. 5.28.

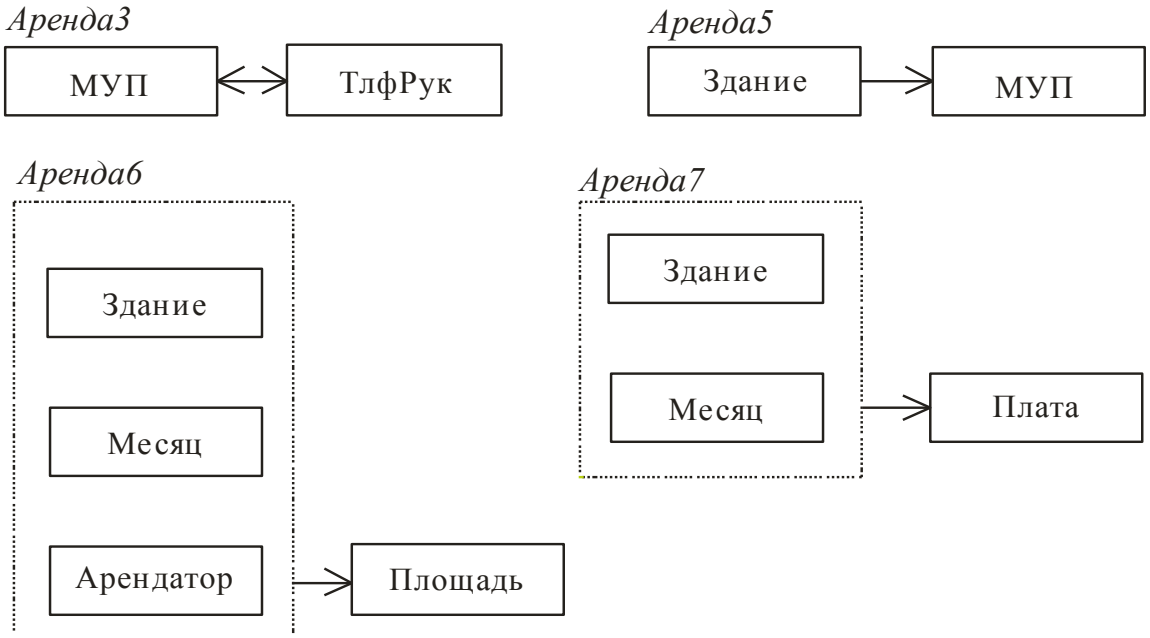


Рис. 5.28. Схема БД *Аренда*

Пример экземпляра базы данных, полученной в соответствии с этой структурой, приводится на рис. 5.29.

Аренда3

МУП	Тлф
Дом быта	123456
Автокомбинат1	234567
Автокомбинат2	345678
Горзеленхоз	121212
Горпроект	123123
Бюро тех. инв.	333333

Аренда5

Здание	МУП
Бабеля 125	Дом быта
Гоголя 2	Автокомбинат1
Бабеля 1	Автокомбинат2
Гаршина 1	Горзеленхоз
Чехова 22	Горпроект
Новая 1	Бюро тех. инв.

Аренда6

Здание	Месяц	Плата
Бабеля 125	200501	1 000
Бабеля 125	200502	1 000
Бабеля 125	200503	1 200
Гоголя 2	200503	500
Бабеля 1	200501	500
Бабеля 1	200502	400
Бабеля 1	200503	400
Гаршина 1	200501	600
Гаршина 1	200502	600
Чехова 22	200501	1 000
Новая 1	200503	900

Аренда7

Здание	Арендатор	Месяц	S
Бабеля 125	Норд	200501	50
Бабеля 125	Норд	200502	50
Бабеля 125	Вест	200501	100
Бабеля 125	Вест	200502	100
Бабеля 125	Вест	200503	150
Гоголя 2	Норд	200503	60
Бабеля 1	АББ	200501	75
Бабеля 1	АББ	200502	75
Бабеля 1	АББ	200503	75
Гаршина 1	Аг. «Travel»	200501	20
Гаршина 1	Аг. «Travel»	200502	20
Чехова 22	АБ «Алемар»	200501	250

Рис. 5.29. Окончательный набор отношений

Чтобы проверить качество спроектированной структуры базы данных *Аренда*, проанализируем выполнение операций добавления, обновления и удаления записей.

Если предположить, что появляется новое муниципальное здание, и помещения в нем еще не сдаются, то общие данные о нем уже могут заноситься

в отношения *Аренда5* и *Аренда7*. Никаких проблем при этом не возникает, все отношения остаются корректными.

При изменении номера телефона в любом МУП достаточно внести соответствующее исправление в единственный кортеж отношения *Аренда3*. Каких-либо противоречий в базе данных не возникнет; нарушение целостности данных не может произойти, поскольку этому препятствует сама структура БД.

Если какой-либо арендатор отказывается от аренды, то достаточно удалить соответствующий кортеж в отношении *Аренда6*. Эти изменения никак не затронут другие отношения и в них сохранятся все данные о зданиях, плате, организациях и т. п.

Таким образом, представление базы данных *Аренда* в виде четырех перечисленных выше отношений является более приемлемым решением, чем ее организация в виде единственного универсального отношения.

Однако следует заметить, что существует возможность определения другой приемлемой схемы БД. Если обратиться к рис. 5.24, то можно заметить, что в универсальном отношении наряду с использованной цепочкой функциональных зависимостей *Здание* $\rightarrow$ *МУП* $\rightarrow$ *Тлф* присутствует цепочка *Здание* $\rightarrow$ *Тлф* $\rightarrow$ *МУП*, которая была проигнорирована. В этой цепочке последней является функциональная зависимость *Тлф* $\rightarrow$ *МУП*. Если бы она была использована для декомпозиции универсального отношения, то мы получили бы базу данных со следующими отношениями:

Аренда3 (Тлф, МУП),
Аренда5 (Здание, Тлф),
Аренда6 (Здание, Арендатор, Месяц, Площадь),
Аренда7 (Здание, Месяц, Плата).

Эта структура БД отличается от предыдущей тем, что в ней атрибут *Тлф* является первичным ключом и поэтому играет более активную и более важную роль, чем атрибут *МУП*. Выбор одного из двух возможных вариантов БД зависит от способа ее использования, следовательно, решающим может стать мнение конечных пользователей по этому вопросу.

Возможность создания двух корректных вариантов базы данных возникла из-за существования *взаимной зависимости* пары атрибутов *МУП* $\leftrightarrow$ *Тлф*. Взаимные зависимости являются причиной появления неоднозначных решений при разработке схемы БД. Признаком взаимной функциональной зависимости атрибутов является значение коэффициента группового отношения, равное 1 : 1.

Обязательным требованием при проектировании структуры отношений является поиск последовательностей функциональных зависимостей и декомпозиция с помощью проекции самой последней функциональной зависимости. Следует всячески избегать проецирования зависимостей, правая часть которых полностью или частично является детерминантом другой функциональной зависимости. Нарушение указанного правила влечет за собой потерю функциональных зависимостей. Чтобы подтвердить это утверждение, рассмотрим следующий пример.

Пусть имеется две функциональных зависимости $X \to Y$ и $Y \to Z$. Отсюда следует функциональная зависимость $X \to Z$. Допустим далее, что отношение $R(X, Y, Z)$ было разбито на два отношения $R_1(X, Z)$ и $R_2(X, Y)$. Предположим, что в какой-то момент времени имелись вполне корректные экземпляры отношений R_1 и R_2 (рис. 5.30).

R_1

X	Z
a	2
b	3
c	5

R_2

X	Y
a	7
b	7
c	7

Рис. 5.30. Пример некорректного разбиения

Если теперь выполнить операцию соединения отношений R_1 и R_2, то получим отношение, представленное на рис. 5.31. В этом отношении легко обнаружить, что связь между Y и Z уже не является функциональной зависимостью, она была утеряна при проектировании схемы БД. Хотя оба отношения R_1 и R_2 находятся в НФБК, при их соединении возникают некорректные связи между Y и Z. Причина их появления – использование функциональной зависимости $X \to Y$, правая часть которой (*Y*) является детерминантом другой функциональной зависимости.

X	Y	Z
a	7	2
b	7	3
c	7	5

Рис. 5.31. Соединение отношений R_1 и R_2

Потеря функциональной зависимости возможна также в случаях, когда один атрибут зависит от двух различных детерминантов. Пусть имеется отношение $R(X, Y, Z)$, в котором *Z* функционально зависит и от *X*, и от *Y* (рис. 5.32).

Данное отношение не находится в нормальной форме Бойса – Кодда, так как детерминантами являются *X* и *Y*, но имеется только один возможный ключ *(X, Y)*. Следовательно, правило цепочек здесь не может быть применено из-за их отсутствия. Более того, если одну из функциональных зависимостей применить обычным способом, то вторая будет утрачена.

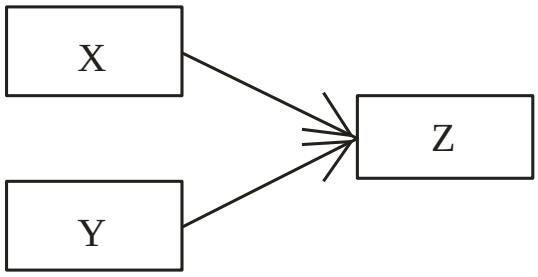


Рис. 5.32. Зависимость от двух детерминантов

В подобных ситуациях проектирование БД методом декомпозиции не дает удовлетворительных результатов, поэтому рекомендуется другой метод, называемый *методом синтеза*. Он основан на объединении всех функциональных зависимостей с одинаковыми детерминантами в группы и

создании для каждой группы отдельного отношения. В соответствии с этим правилом в последнем примере следует создать два отношения $R_1(X, Z)$ и $R_2(Y, Z)$.

Метод декомпозиции может рассматриваться как основной, а метод синтеза – как дополнение к нему в тех случаях, когда применение метода декомпозиции сопряжено с проблемами.

5.14. Модификации метода декомпозиции

Применение метода декомпозиции сопряжено и с другими проблемами. Одна из таких проблем возникает при наличии избыточных функциональных зависимостей. Функциональная зависимость называется *избыточной*, если содержит в себе информацию, которая может быть получена из других функциональных зависимостей, уже использованных при проектировании БД. Но поскольку избыточные зависимости не содержат дополнительной информации, они могут быть удалены без какого-либо ущерба для корректного отображения предметной области. Удаление избыточных функциональных зависимостей должно выполняться до начала процесса декомпозиции.

Одной из причин возникновения избыточных зависимостей является свойство их транзитивности. Если имеются функциональные зависимости $X \to Y$ и $Y \to Z$, то существует зависимость $X \to Z$, называемая *транзитивной*. Транзитивные зависимости являются вполне корректными, но их использование для разбиения отношений не требуется. Следовательно, из трех отношений $R_1(X, Y)$, $R_2(Y, Z)$ и $R_3(X, Z)$ последнее является избыточным и должно быть удалено из схемы базы данных.

На рис. 5.33 показан пример исключения транзитивных зависимостей. В исходной схеме БД присутствовали следующие функциональные зависимости: $W \to X$, $W \to Y$, $W \to Z$, $X \to Y$, $X \to Z$, $Y \to Z$. В результате исключения избыточных зависимостей получена следующая последовательность функциональных зависимостей: $W \to X \to Y \to Z$. Выполнив ее декомпозицию по правилу цепочек зависимостей, получим такой набор отношений: $R_1(Y, Z)$, $R_2(X, Y)$, $R_3(W, X)$.

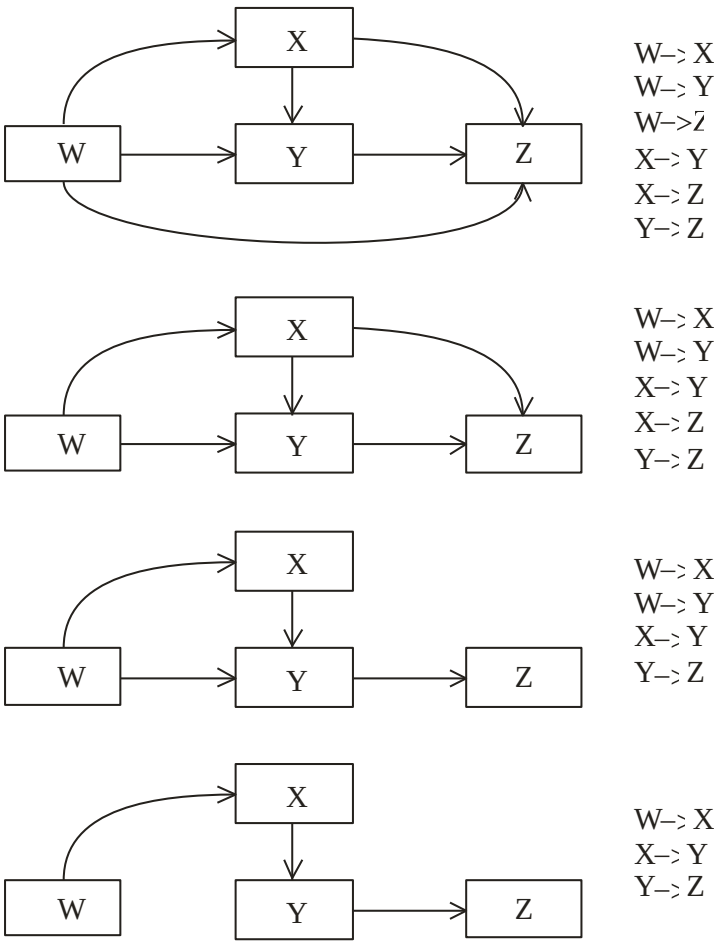


Рис. 5.33. Исключение транзитивных зависимостей

Другой причиной появления избыточных функциональных зависимостей служит добавление атрибутов в существующее отношение. Такое добавление имеет несколько разновидностей. Пусть X, Y и Z – некоторые атрибуты, каждый из которых может быть составным. Тогда первый вид подобной функциональной избыточности определяют так: если $X \to Z$, то $(X,Y) \to Z$ также корректная, но избыточная функциональная зависимость (рис. 5.34).

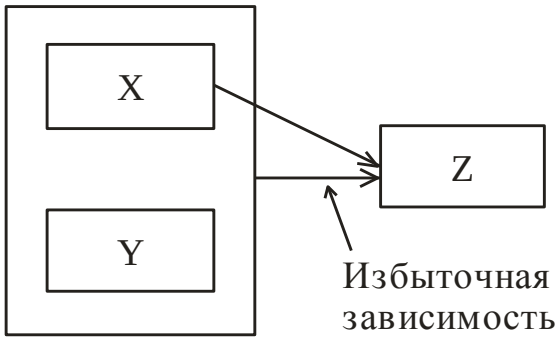


Рис. 5.34. Пример добавления 1

Вторая разновидность избыточной зависимости возникает при создании новой функциональной зависимости путем добавления одного и того же атрибута к обеим частям уже существующей зависимости: если $X \to Z$, то $(X,Y) \to (Z,Y)$ – корректная, но избыточная зависимость (рис. 5.35).

Модификация исходных функциональных зависимостей с целью получения других, эквивалентных им зависимостей может осуществляться с помощью трех правил вывода: объединения, декомпозиции и псевдотранзитивности.

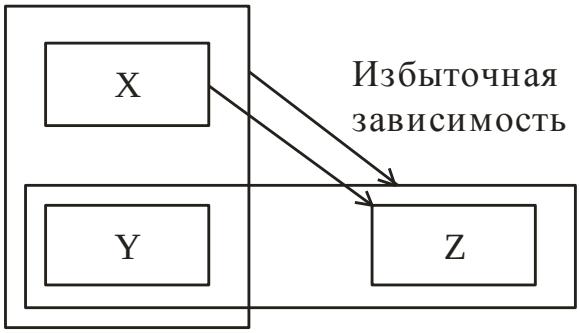


Рис. 5.35. Пример добавления 2

Правило объединения формулируется следующим образом: если $X \to Y$ и $X \to Z$, то корректной будет функциональная зависимость $X \to (Y, Z)$ (рис. 5.36).

Правило декомпозиции является обратным по отношению к правилу объединения: если $X \to (Y, Z)$, то $X \to Y$ и $X \to Z$ (рис. 5.37).

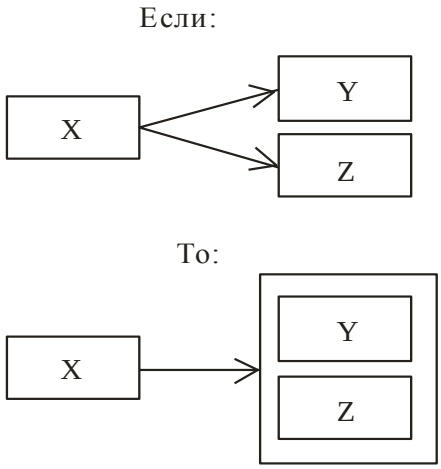


Рис. 5.36. Правило объединения

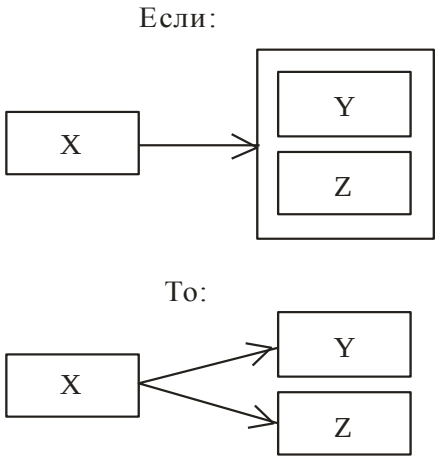


Рис. 5.37. Правило декомпозиции

Избыточность по причине псевдотранзитивности возникает тогда, когда в получаемых функциональных зависимостях присутствуют составные детерминанты. Если $X \to Y$ и $(Y, W) \to Z$, то зависимость $(X, W) \to Z$ избыточна в силу псевдотранзитивности (рис. 5.38).

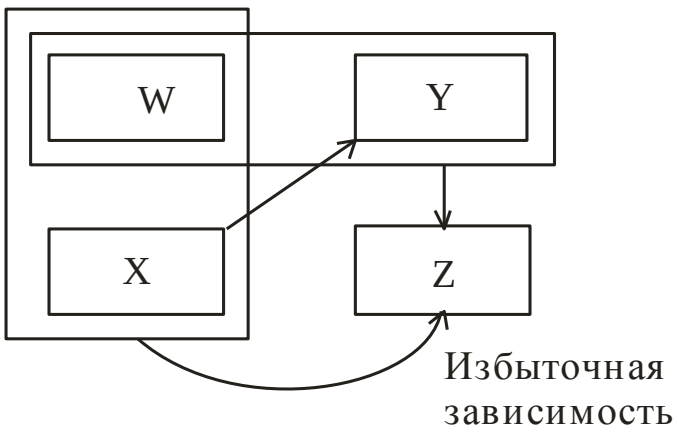


Рис. 5.38. Пример псевдотранзитивности

Приведенные выше правила являются следствием аксиом, получивших название *аксиом Армстронга* по имени их автора.

1. Рефлексивность: если Y является подмножеством X, то $X \to Y$.

2. Дополнение: если $X \to Y$, то $X \cup Z \to Y \cup Z$.

3. Транзитивность: если $X \to Y$ и $Y \to Z$, то $X \to Z$.
4. Самоопределение: $X \to X$.
5. Декомпозиция: если $X \to YZ$, то $X \to Y$ и $X \to Z$.
6. Объединение: если $X \to Y$ и $X \to Z$, то $X \to YZ$.
7. Композиция: если $X \to Y$ и $V \to W$, то $X \bigcup V \to Y \bigcup W$.

Перечисленные аксиомы дополняются *теоремой всеобщего объединения*: если $X \to Y$ и $V \to W$, то $X \bigcup (V - Y) \to Y \bigcup W$. Данная теорема, доказанная Дарвеном, названа так потому, что некоторые перечисленные выше правила могут быть получены как ее частные случаи.

Совокупность функциональных зависимостей, полученную из исходной удалением всех избыточных зависимостей, называют *минимальным покрытием.* Минимальное покрытие может быть не единственным. В зависимости от порядка удаления избыточных функциональных зависимостей, можно получить альтернативные наборы зависимостей, каждый из которых является минимальным покрытием.

Описанный выше алгоритм проектирования структуры БД теперь может быть уточнен и представлен как последовательность следующих пяти этапов.

1. Представление БД в виде универсального отношения.
2. Определение всех функциональных зависимостей между атрибутами универсального отношения.
3. Получение минимального покрытия путем последовательного удаления избыточных функциональных зависимостей и анализ получаемого набора зависимостей на наличие избыточных.
4. Создание набора отношений в нормальной форме Бойса – Кодда методом декомпозиции универсального отношения с использованием различных вариантов минимального покрытия.
5. Сравнение баз данных, полученных на основе различных минимальных покрытий, и выбор наилучшего варианта.

Если в процессе декомпозиции дальнейшее проецирование без потерь функциональных зависимостей становится невозможным, то необходимо:

– создать отдельное отношение для каждого детерминанта и набора зависящих от него атрибутов;

– либо изменить порядок выполненных ранее декомпозиций.

По окончании разработки структуры базы данных все множество полученных отношений должно быть проанализировано с целью выявления возможных проблем. Для этого необходимо составить перечень всех функциональных зависимостей. Полученные функциональные зависимости должны отвечать двум требованиям:

– каждая функциональная зависимость должна присутствовать не более чем в одном отношении;

– набор функциональных зависимостей, полученных в результате проектирования, должен полностью совпадать с минимальным покрытием. Если это не так, то необходимо показать с помощью правил вывода

возможность получения полученного набора зависимостей из минимального покрытия.

Кроме того, полученный набор отношений должен быть проверен на предмет избыточности. Отношение является избыточным, если

– все атрибуты этого отношения полностью содержатся в каком-либо другом отношении;

– все атрибуты избыточного отношения содержатся в отношении, которое может быть получено из других отношений с помощью операции их соединения.

Все избыточные отношения должны быть удалены из конечного набора отношений.

Последней процедурой должна быть проверка базы данных с практической точки зрения. Разработчик должен оценить, будет ли БД поддерживать типичные виды запросов и предполагаемые операции ее обновления.

5.15. Сущности и связи

Отношения можно трактовать в некотором смысле как объекты. Если объект определить как нечто, о чем можно хранить информацию, то отношение, понимаемое как некоторая связь между объектами, также будет отвечать этому определению. Один и тот же элемент реального мира разными пользователями может рассматриваться как объект, как свойство и как отношение. Рассмотрим в качестве примера поставку различных продуктов. Очевидно, что каждый продукт обладает определенными свойствами и поэтому при необходимости (или в зависимости от целей базы данных) может рассматриваться в качестве объекта. Если никакие свойства продукта пользователя не интересуют, а важно только то, кто его поставляет, то продукт можно рассматривать как свойство объекта «поставщик». Если рассматривать пару «поставщик – потребитель», то продукт может трактоваться как отношение, поскольку связь между поставщиками и потребителями возникает в силу существования поставки продукта.

Недостаток реляционных и любых других СУБД состоит в том, что они обладают крайне ограниченными сведениями о *смысловом значении* данных и по этой причине не в состоянии понимать вопросы пользователя на *семантическом уровне*. Поэтому в области баз данных возникло направление, называемое *семантическим моделированием*. Основной целью семантического моделирования является обеспечение гибкости при интерпретации тех или иных данных (символов) в ситуациях, подобных описанной в нашем последнем примере.

Пока что можно сказать, что с семантическим моделированием связано больше надежд, чем осязаемых результатов. По этому поводу К.Дж. Дейт замечал, что основы семантического моделирования не являются такими строгими и ясными, как принципы нормализации. По Дейту, причина этого факта заключается в том, что «…проектирование баз данных все еще является не объективным, а весьма субъективным занятием, поскольку существует сравнительно мало действительно строгих принципов, которые могут

использоваться для разрешения этой проблемы (немногие существующие на сегодняшний день принципы в основном являются принципами нормализации)» [1, с. 346].

В [2] не без оснований отмечается, что существование различных методов проектирования, которые могут использоваться как в отдельности, так и совместно, свидетельствует о том, что проектирование структуры БД продолжает оставаться частично наукой, а частично – искусством.

Тем не менее, некоторые идеи семантического моделирования оказываются полезными при проектировании баз данных даже тогда, когда используемые СУБД не оказывают непосредственной поддержки этих идей. Одна из концепций семантического моделирования состоит в разделении всех используемых понятий предметной области на сущности (или объекты) и отношения.

Рассмотренный выше декомпозиционный метод проектирования БД рекомендуется к применению при малом (15–20) числе атрибутов. При существенно большем числе атрибутов его применение становится затруднительным из-за громоздкости, и тогда используются другие методы. Одним из таких методов является ER-метод, или метод «сущность – связь» (Entity – Relation).

Сущностью называют некоторый тип объектов (предметов или процессов и т. п.) предметной области, подлежащих отображению в базе данных. В предметной области каждый тип объектов имеет отдельные экземпляры, отличающиеся друг от друга и однозначно идентифицируемые. Единственный признак, который может использоваться для нахождения сущностей, состоит в том, что сущность, как правило, определяется существительным. Примерами сущностей могут служить личности, геометрические и семантические объекты, территории, предприятия, документы, искусственные или естественные объекты географического пространства, абстрактные объекты и т. п.

Связь определяется как некоторое взаимодействие между двумя или более сущностями. Чаще всего связи обозначаются глаголами.

Атрибутами называют отображаемые свойства сущностей. Как правило, одни и те же объекты в разных базах данных характеризуются различными наборами атрибутов. И тогда одни и те же объекты рассматриваются как разные сущности. Например, такой объект, как человек, на предприятии может рассматриваться как сущность «работник», в медицинском учреждении – как «пациент», в военкомате – как «военнослужащий», в органах внутренних дел – как «гражданин» и т. д.

Если объект понимать как то, о чем можно хранить информацию, то эта информация представляет собой некоторые характеристики или свойства объекта. Иными словами, свойства представляют собой те сведения, что хранятся об объектах. Как и объекты, свойства могут иметь сложную структуру.

При графическом представлении взаимосвязей между сущностями, атрибутами и связями различают *диаграммы ER-экземпляров* и *диаграммы ER-типов*. На диаграмме ER-экземпляров указываются связи между индивидными экземплярами тех или иных сущностей. На рис. 5.39, в частности, указываются

государства, столицей которых являются конкретные западноевропейские города. Названия сущностей указываются прописными буквами над экземплярами этих сущностей. Экземпляры сущностей идентифицируются значением атрибутов. Так, ГОРОД – это название сущности, а Афины – конкретный экземпляр сущности (индивидный объект). Экземпляры связей специфицируются линиями между двумя сущностями и также именуются прописными буквами. Имена связей размещаются над экземплярами связей. На рис. 5.39 связь между экземпляром Афины сущности ГОРОД и экземпляром Греция сущности СТРАНА отображает тот факт, что Афины являются столицей Греции.

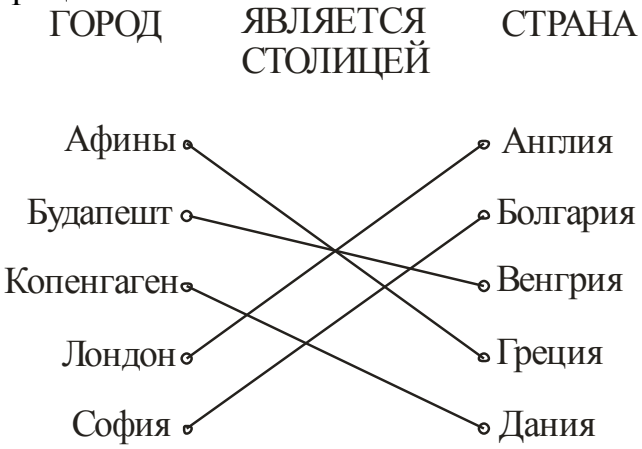


Рис. 5.39. Пример диаграммы ER-экземпляров

В некоторых случаях для идентификации каждого экземпляра сущности может потребоваться набор атрибутов. Такой атрибут или набор атрибутов, используемый для однозначной идентификации экземпляров сущностей, называют *ключом сущности.* Тогда каждый экземпляр связи однозначно идентифицируется набором ключей сущностей, соединяемых этой связью; набор таких ключей называют *ключом связи*.

Диаграммы ER-типов (рис. 5.40) содержат обобщенную информацию, представленную в диаграммах ER-экземпляров. Но хотя эта информация обобщена, она достаточна для проектирования структуры баз данных. На диаграммах ER-типов сущности обозначаются прямоугольниками, внутри которых указываются имена сущностей. Ниже каждой сущности перечисляются имена атрибутов, составляющих ее ключ. Для обозначения связей используются ромбы с именами связей внутри. Важной характеристикой связи служит ее степень. На рис. 5.40 степень связи равна 1 : 1; это означает, что одному экземпляру сущности ГОРОД соответствует один экземпляр сущности СТРАНА и, наоборот, с одним экземпляром сущности СТРАНА связан только один экземпляр сущности ГОРОД.

Рис. 5.40. Пример диаграммы ER-типа

Другим свойством связей между сущностями является *класс принадлежности*. Рассмотрим данное свойство связей на следующем примере.

Допустим, имеется транспортная фирма «Альфа», сдающая в аренду автомобили и имеющая сложившийся круг постоянных клиентов. Допустим далее, что принято решение создать базу данных, в которой предполагается хранить сведения о распределении автомобилей между организациями. Также предположим, что каждый автомобиль может обслуживать только одну организацию и каждая организация может арендовать только один автомобиль. Тогда возможны следующие четыре варианта, представленные на рис. 5.41 с помощью диаграммы ER-экземпляров. Рис. 5.41, *а* отображает ситуацию, когда наличие обеих сущностей в связи является обязательным. Это следует понимать таким образом, что каждый автомобиль обслуживает только одну организацию, и каждая организация арендует ровно один автомобиль.

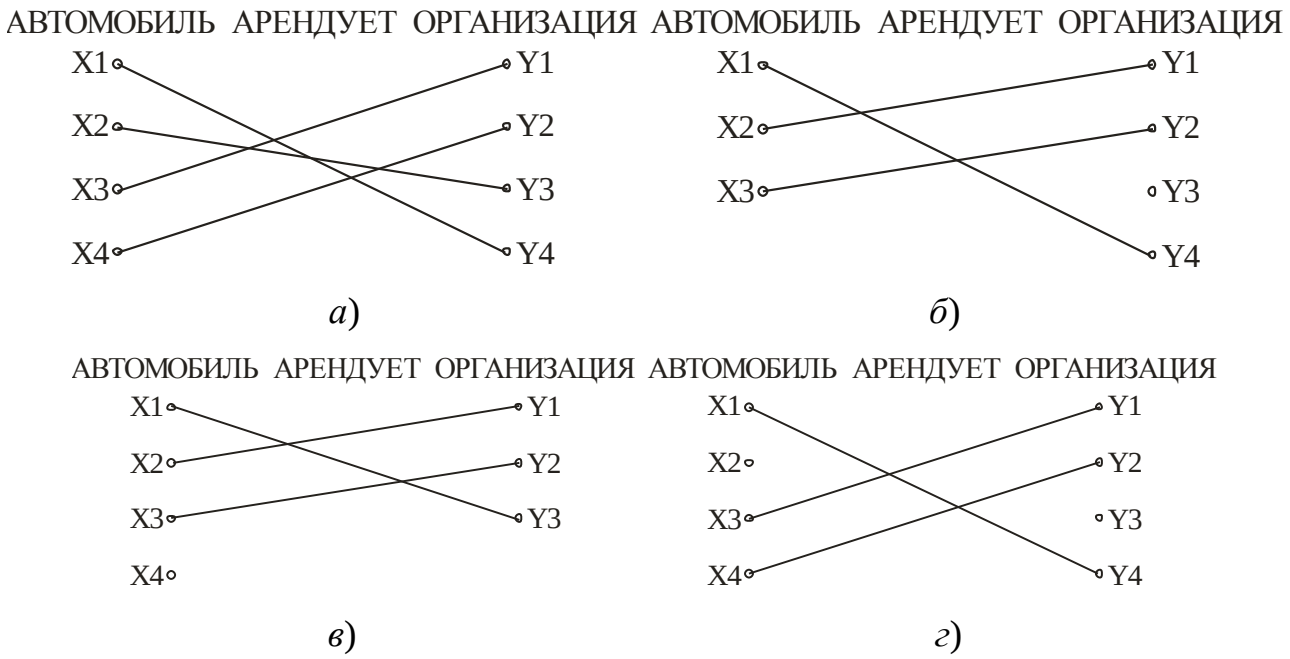


Рис. 5.41. Классы принадлежности

На рис. 5.41, *б* отображена ситуация, когда в связи АРЕНДУЕТ присутствие второй сущности не является обязательным. Например, когда автомобилей не хватает для обслуживания всех организаций, то есть организаций больше, чем автомобилей. Рис. 5.41, *в* соответствует случаю, когда в связи между сущностями первая сущность может отсутствовать (когда автомобилей больше, чем организаций). На рис. 5.41, *г* в связи могут не присутствовать обе сущности. Это будет в том случае, когда некоторые организации не подают заявки, и некоторые автомобили не арендуются.

Данное свойство связей называется *классом принадлежности* и обозначает обязательность или необязательность участия сущностей в связи. По этому свойству будем называть участие сущностей *обязательным* или *необязательным.* Если участие сущности в некоторой связи является обязательным, то на диаграммах ER-типов этот факт отображается с помощью небольшого прямоугольника, примыкающего к прямоугольнику соответствующей сущности. В противном случае такой прямоугольник отсутствует (рис. 5.42, *б, в, г*).

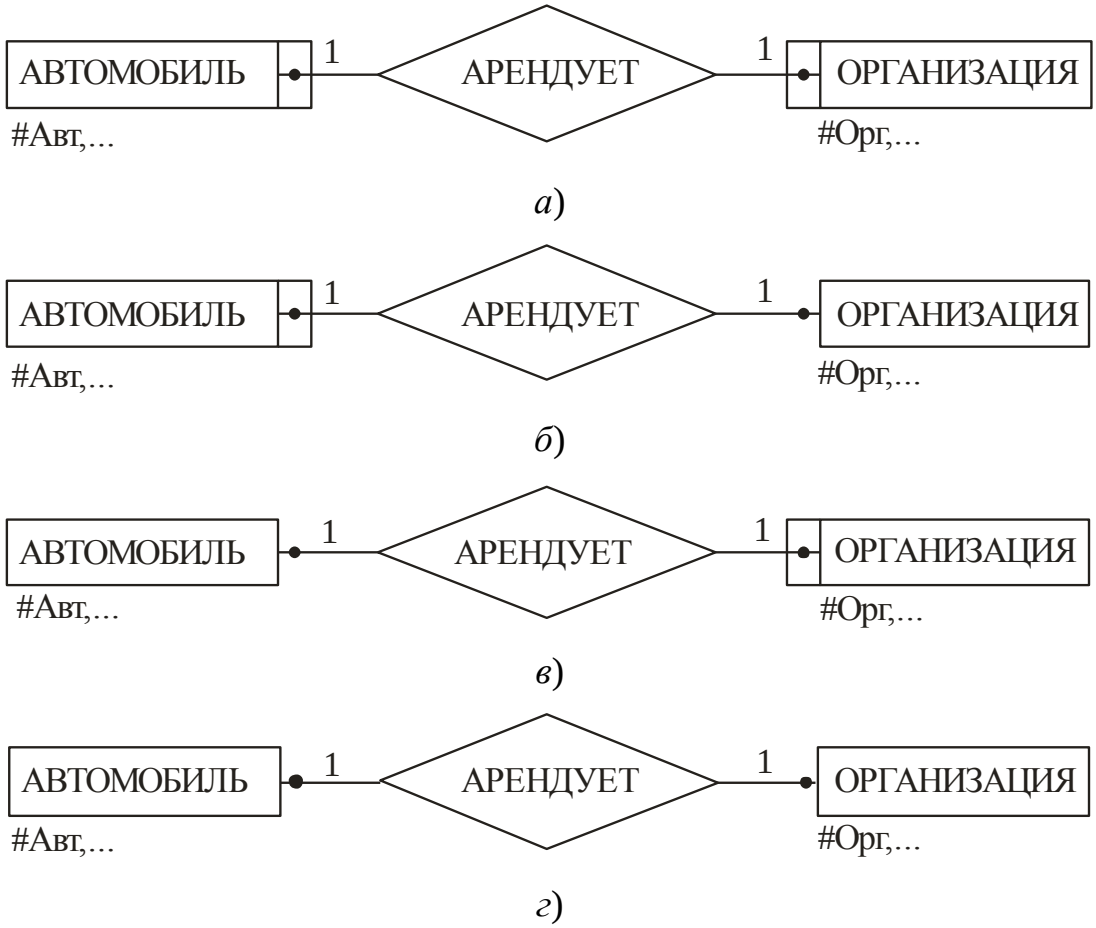


Рис. 5.42. Классы принадлежности для связи степени 1 : 1

Если в связи между сущностями экземпляру второй сущности может быть поставлено в соответствие более одного экземпляра первой сущности, то говорят, что степень такой связи равна *n* : 1 (рис. 5.43).

Предположим, что другая транспортная фирма «Бетта» работает в других условиях, представляет автомобили более крупным заказчикам и для удовлетворения потребностей в перевозках некоторым из них требуется более одного автомобиля (см. рис. 5.43). Рис. 5.43, *а* соответствует случаю, когда каждый автомобиль обслуживает одну организацию и любая организация арендует один или несколько автомобилей. Рис. 5.43, *б* следует понимать так, что каждая автомашина арендуется одной организацией, но одни из них могут не арендовать автомобили вообще, а другие могут арендовать несколько машин.

На рис. 5.43, *в* представлен случай, когда каждая организация арендует один или более автомобилей, но некоторые автомобили не арендованы ни одной организацией, то есть сущность АВТОМОБИЛЬ является необязательной. Рис. 5.43, *г* описывает ситуацию, когда каждая организация арендует 0, 1 или более автомобилей, а каждый автомобиль обслуживает только одну организацию, либо не используется вообще.

а)

б)

в)

г)

Рис. 5.43. Классы принадлежности для связи степени n : 1

Если в связи между сущностями экземпляру первой сущности может соответствовать несколько экземпляров второй сущности, то говорят, что степень такой связи равна 1 : *n* (рис. 5.44). Предположим, что транспортная фирма «Гамма» обслуживает множество мелких организаций и использование навигационной ГИС и GPS позволяет ей оптимизировать выполнение заказов таким образом, что каждый автомобиль из некоторого их числа может обслужить нескольких заказчиков.

Диаграмма ER-типов на рис. 5.44, *а* соответствует случаю, когда автомобиль может выполнить несколько заказов, и каждый заказчик арендует один автомобиль. На рис. 5.44, *б* представлен случай, когда автомобиль обслуживает несколько организаций, но каждая организация арендует 0 или 1 автомобиль.

Рис. 5.44, *в* описывает ситуацию, когда каждая организация арендует один автомобиль, один автомобиль может обслужить несколько организаций, но некоторые автомобили не арендуются. Рис. 5.44, *г* соответствует случаю, когда любая организация может арендовать не более одного автомобиля (то есть некоторые могут не арендовать), и одни автомобили могут обслужить несколько организаций, а другие автомобили могут оказаться излишними.

а)

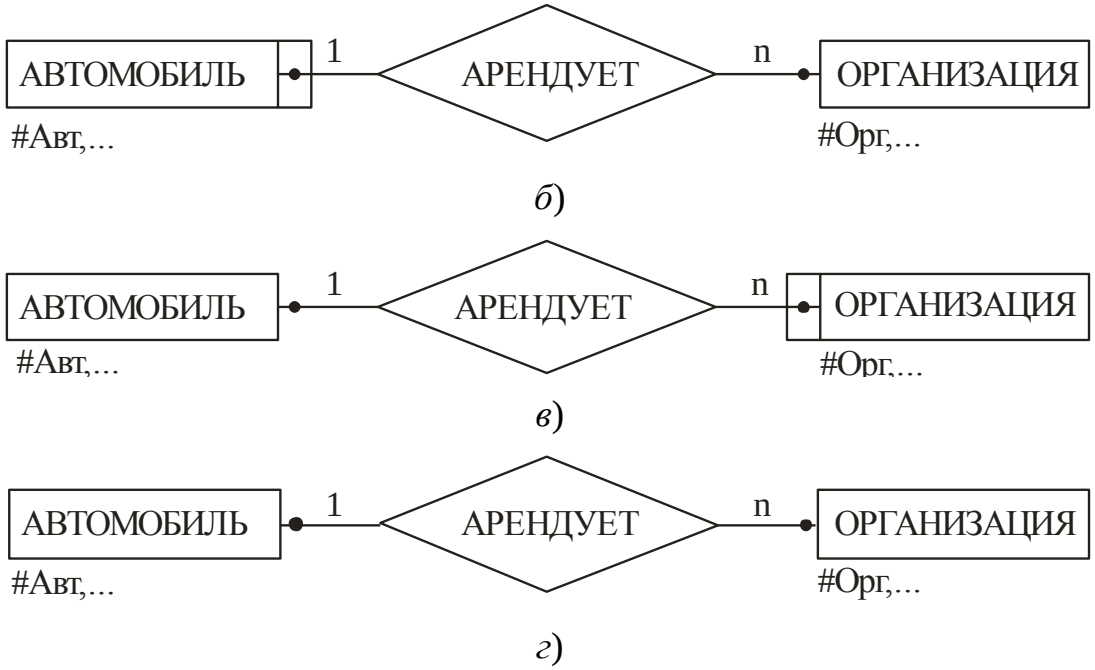
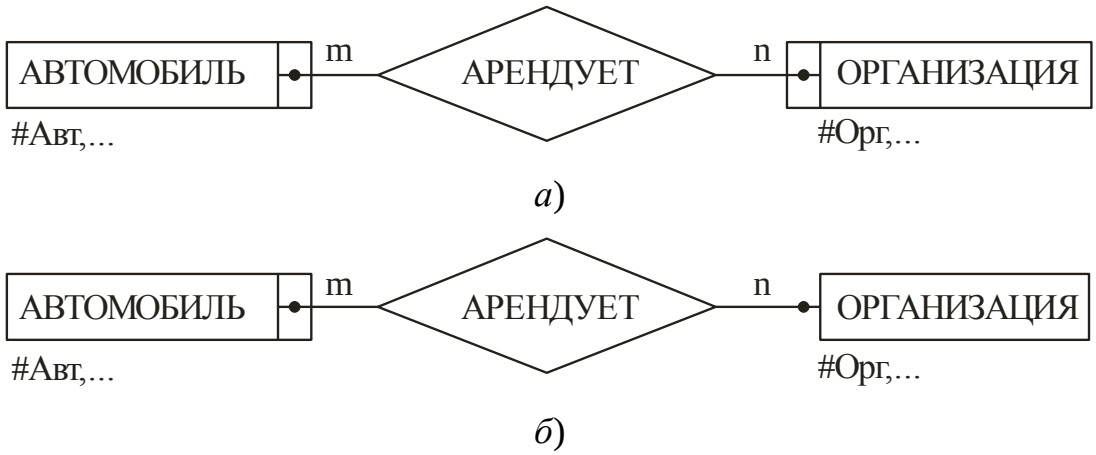


Рис. 5.44. Классы принадлежности для связи степени 1 : n

Предыдущие примеры были в определенной мере искусственными. Пример на рис. 5.45 носит более реальный характер. Предположим, что требуется создать базу данных в крупной транспортной фирме «Дельта». Рис. 5.45, *а* отображает идеальную ситуацию, когда каждая организация заказывает один или более автомобилей, а каждый автомобиль обслуживает не менее чем одну организацию. Рис. 5.45, *б* соответствует ситуации, когда все автомобили арендованы, и некоторые из них обслуживают несколько организаций, но не все организации арендуют автомобили. На рис. 5.45, *в* показан случай, когда каждая организация арендует некоторое число автомобилей, но не все автомобили арендованы. Наконец, рис. 5.45, *г* следует трактовать так, что каждая организация арендует 0, 1 или $m > 1$ автомобилей, а каждый автомобиль обслуживает 0, 1 или $n > 1$ организаций. Если бы возникла необходимость разрабатывать подобную базу данных в реальной жизни, то последний вариант следовало бы выбрать в качестве наиболее вероятного.



в)

г)

Рис. 5.45. Классы принадлежности для связи степени m : n

5.16. Построение отношений по ER-диаграммам

Рассмотренные выше связи называют *бинарными*, поскольку в них участвует две сущности. Бинарные связи являются наиболее распространенными, хотя встречаются связи более высокого порядка, в которых участвует более двух сущностей. Наша ближайшая задача состоит в получении отношений по ER-диаграммам.

Решение поставленной задачи состоит в выполнении нескольких этапов. На первом этапе необходимо построить ER-диаграммы, содержащие все сущности и все связи в конкретной предметной области. Второй этап состоит в построении набора предварительных отношений и определении первичного ключа для каждого полученного отношения. Третий этап заключается в подготовке списка всех атрибутов и назначении каждому атрибуту одного из предварительных отношений таким образом, чтобы эти отношения находились в НФБК. В процессе выполнения этого этапа определяются межатрибутные функциональные зависимости, с помощью которых выполняется проверка принадлежности отношений нормальной форме Бойса – Кодда. Если полученное множество отношений не находится в НФБК или существуют затруднения в распределении атрибутов по отношениям, то необходимо подвергнуть ревизии ER-диаграммы с целью устранения возникших проблем.

Предварительный набор отношений для конкретной бинарной связи может быть получен в результате рассмотрения возможных альтернатив и выбора из их числа наиболее адекватной реальным условиям. Перечисляемые ниже общие правила создания отношений из ER-диаграмм бинарных связей основываются на анализе степени связи и классе принадлежности.

Если степень бинарной связи равна 1 : 1 и обе сущности являются обязательными, то достаточно одного отношения, первичным ключом которого может быть ключ любой из двух сущностей (см. рис. 5.41, *а*). Выбор ключа осуществляется с учетом того, каким образом будет преимущественно использоваться БД.

Продолжая пример с транспортной фирмой, предположим, что в ее распоряжении имеются автомобили, перечисленные на рис. 5.46, и она обслуживает фирмы, указанные на рис. 5.47.

#авт	Марка
1	Волга
2	Волга
3	Хонда
4	Лада
5	Нива

Рис. 5.46. Перечень автомобилей

#орг	Название	Телефон
1	Астра	123456
2	Вега	222222
3	Полюс	123123
4	Восток	232323
5	Сибирь	121212

Рис. 5.47. Перечень организаций

Допустим далее, что сущность АВТОМОБИЛЬ характеризуется атрибутами #авт, марка и т. д. Атрибут #авт (целое число) является уникальным и может служить ключом сущности АВТОМОБИЛЬ. Сущность ОРГАНИЗАЦИЯ представляется атрибутами #орг, название и телефон. Атрибуты #орг (целое число) и телефон (целое число) являются уникальными, в качестве ключа сущности ОРГАНИЗАЦИЯ выбран атрибут #орг.

Тогда в соответствии с указанным выше правилом возможно создание БД в виде одного из двух отношений:

АЛЬФА1 (#авт, марка, #орг, назв, тлф);

АЛЬФА2 (#орг, назв, тлф, #авт, марка).

Данные отношения отличаются только выбором первичного ключа. Экземпляр первого отношения мог бы иметь вид, представленный на рис. 5.48.

АЛЬФА1

#авт	марка	# орг	назв	тлф
1	Волга	4	Восток	232323
2	Волга	2	Вега	222222
3	Хонда	1	Астра	123456
4	Лада	5	Сибирь	121212
5	Нива	3	Полюс	123123

Рис. 5.48. Экземпляр БД АЛЬФА1

Если степень бинарной связи равна 1 : 1 и одна из сущностей является необязательной, то одного отношения уже недостаточно. Если мы все-таки создадим такое отношение, то в нем могут возникнуть незаполненные поля, что легко проверить хотя бы на примере, приведенном на рис. 5.41, *б* и 5.41, *в*.

Поэтому, *если степень бинарной связи равна 1 : 1 и только одна из сущностей является необязательной, то необходимо два отношения. При этом для каждой сущности создается одно отношение с ключом этой сущности в качестве первичного ключа. Кроме того, ключ необязательной сущности добавляется в качестве атрибута в отношение, созданное для обязательной сущности.*

Если в нашем примере необязательной сущностью является ОРГАНИЗАЦИЯ (см. рис. 5.41, *б*), то в соответствии с данным правилом должны быть созданы два отношения:

АВТОМОБИЛЬ (#авт, марка, #орг);

ОРГАНИЗАЦИЯ (#орг, назв, тлф).

Тогда экземпляры отношений в некоторый момент времени могли бы выглядеть так, как представлено на рис. 5.49.

АВТОМОБИЛИ

#авт	марка	# орг
1	Волга	4
2	Волга	3
3	Хонда	1
4	Лада	2

ОРГАНИЗАЦИИ

#орг	назв	тлф
1	Астра	123456
2	Вега	222222
3	Полюс	123123
4	Восток	232323
5	Сибирь	121212

Рис. 5.49. Экземпляр БД АЛЬФА2

Если необязательной сущностью является АВТОМОБИЛЬ (см. рис. 5.41, *в*), то необходимо создать отношения:

АВТОМОБИЛЬ (#авт, марка);

ОРГАНИЗАЦИЯ (#орг, назв, тлф, #авт).

Если степень бинарной связи равна 1 : 1 и обе сущности не являются обязательными, то необходимо создать три отношения: по одному для каждой сущности, ключи которых являются первичными ключами соответствующих отношений, и одного отношения для связи, в которое включаются в качестве атрибутов ключи обеих сущностей.

Таким образом, для случая на рис. 5.41, *г* следует создать три отношения:

АВТОМОБИЛЬ (#авт, марка);

ОРГАНИЗАЦИЯ (#орг, назв, тлф);

ЗАКАЗ (#авт, #орг).

В некоторый момент времени экземпляры отношений могли бы выглядеть так, как представлено на рис. 5.50.

АВТОМОБИЛИ

#авт	марка
1	Волга
2	Волга
3	Хонда
4	Лада
5	Нива

ОРГАНИЗАЦИИ

#орг	назв	тлф
1	Астра	123456
2	Вега	222222
3	Полюс	123123
4	Восток	232323
5	Сибирь	121212

АРЕНДА

#авт	# орг
1	3
3	2
4	1
5	4

Рис. 5.50. Экземпляр БД АЛЬФА3

Для бинарных связей степени 1 : *n* основным фактором, определяющим структуру отношений, является класс принадлежности *n*-связной сущности; класс принадлежности 1-связной сущности на нее никаким образом не влияет. Поэтому для связей степени 1 : *n* устанавливаются всего два правила.

Если степень бинарной связи равна 1 : n и n-связная сущность является обязательной, то необходимо использовать два отношения, по одному для каждой сущности, при условии, что ключ сущности используется в качестве первичного ключа соответствующего отношения; кроме того, ключ 1-связной

сущности должен быть добавлен в отношение, соответствующее n-связной сущности.

Для случая, изображенного на рис. 5.44, *а*, *в*, необходимо создать два отношения:

АВТОМОБИЛЬ (#авт, марка);

ОРГАНИЗАЦИЯ (#орг, назв, тлф, #авт).

Примером экземпляров отношений для бинарной связи 1 : *n*, в которой *n*-связная сущность является обязательной, может служить рис. 5.51.

АВТОМОБИЛИ

#авт	марка
1	Волга
2	Волга
3	Хонда
4	Лада
5	Нива

ОРГАНИЗАЦИИ

#орг	назв	тлф	#авт
1	Астра	123456	2
2	Вега	222222	2
3	Полюс	123123	4
4	Восток	232323	3
5	Сибирь	121212	3

Рис. 5.51. Экземпляр БД ГАММА1

Если степень бинарной связи равна 1 : n и n-связная сущность является необязательной, то необходимо создать три отношения: по одному для каждой сущности (с ключом этой сущности в качестве первичного ключа соответствующего отношения) и одного отношения для связи, в число атрибутов которого должны быть помещены ключи каждой сущности.

Следовательно, для случая, представленного на рис. 5.44, б, г, требуется создание следующих отношений:

АВТОМОБИЛЬ (#авт, марка);

ОРГАНИЗАЦИЯ (#орг, назв, тлф);

АРЕНДА (#авт, #орг).

Возможный экземпляр базы данных представлен на рис. 5.52.

АВТОМОБИЛИ

#авт	марка
1	Волга
2	Волга
3	Хонда
4	Лада
5	Нива

ОРГАНИЗАЦИИ

#орг	назв	тлф
1	Астра	123456
2	Вега	222222
3	Полюс	123123
4	Восток	232323
5	Сибирь	121212

АРЕНДА

#авт	#орг
2	1
2	2
3	4
4	3

Рис. 5.52. Экземпляр БД ГАММА2

Разработка отношений для связей со степенью *n* : 1 сводится к случаю связей со степенью 1 : *n* перестановкой сущностей в связи.

Число отношений для бинарной связи со степенью *m* : *n* не зависит от класса принадлежности как первой, так и второй сущности, поэтому используется следующее правило.

Если степень бинарной связи равна m : n, то для представления данных требуется три отношения: по одному отношению для каждой сущности, в

которых первичными ключами служат ключи сущностей, и одного отношения для связи, среди атрибутов которого должны присутствовать ключи обеих сущностей.

В соответствии с данным правилом для случая, представленного на рис. 5.45, следует создать три отношения:

АВТОМОБИЛЬ (#авт, марка);

ОРГАНИЗАЦИЯ (#орг, назв, тлф);

АРЕНДА (#авт, #орг).

Примеры экземпляров отношений для бинарной связи сущностей со степенью связи, равной *m*:*n*, приведены на рис. 5.53.

АВТОМОБИЛИ

#авт	марка
1	Волга
2	Волга
3	Хонда
4	Лада
5	Нива

ОРГАНИЗАЦИИ

#орг	назв	тлф
1	Астра	123456
2	Вега	222222
3	Полюс	123123
4	Восток	232323
5	Сибирь	121212

АРЕНДА

#авт	#орг
2	1
2	2
3	4
4	3

Рис. 5.53. Экземпляр БД ДЕЛЬТА

В литературе можно встретить рекомендации создавать по одному отношению для каждой сущности и одному отношению для каждой связи. Однако подобное решение обычно влечет за собой создание большего числа отношений, чем это необходимо.

5.17. Связи более высокого порядка

Выше были рассмотрены ситуации, когда для построения информационной модели предметной области использовались бинарные связи между сущностями. Хотя с их помощью можно адекватно отобразить многие стороны реального мира, в некоторых случаях требуется применение более сложных конструкций – связей более высокого порядка.

Рассмотрим связи более высокого порядка на следующем безрадостном примере. Предположим, что в некотором медицинском учреждении решили изучить эффективность лечения в разных санаториях и с этой целью создать соответствующую базу данных.

В данном примере можно выделить сущности ПАЦИЕНТ, БОЛЕЗНЬ и САНАТОРИЙ. Сущность ПАЦИЕНТ характеризуется такими атрибутами, как уникальный номер карточки пациента (#Пцнт), фамилия, имя и отчество (ФИО), год рождения (Гржд) и другими, которые мы для краткости опустим. Сущность БОЛЕЗНЬ имеет такие атрибуты, как уникальный код болезни (#Бол), название болезни (БНзв), симптомы и т. д. Для описания сущности САНАТОРИЙ используются атрибуты: уникальный код санатория (#Снтр), название (СНзв), ближайший город (Грд).

Рассмотрим связи между перечисленными сущностями, но первоначально решение поставленной задачи рассмотрим в упрощенном виде. Допустим, что

каждый пациент страдает только одной болезнью. Поэтому связь между сущностями ПАЦИЕНТ и БОЛЕЗНЬ, которую мы назовем БОЛЕЕТ, имеет степень *n* : 1 (одним заболеванием могут страдать разные пациенты) и *n*-связная сущность ПАЦИЕНТ в этой связи по условию является обязательной (у каждого пациента есть какая-либо болезнь).

Связь ЛЕЧАТ между сущностями САНАТОРИЙ и БОЛЕЗНЬ означает, что в определенном санатории проводится лечение тех или иных болезней. В одном санатории может осуществляться лечение нескольких болезней, а лечением одной и той же болезни могут заниматься разные санатории. Поэтому связь ЛЕЧАТ имеет степень *m* : *n*. Будем считать, что лечением любой болезни занимаются хотя бы в одном санатории, а в каждом санатории лечат хотя бы одно заболевание. Следовательно, обе сущности в связи ЛЕЧАТ являются обязательными.

Связь ЛЕЧИТСЯ между сущностями ПАЦИЕНТ и САНАТОРИЙ свидетельствует о том, кто из больных и в каком санатории проходил лечение. Будем предполагать, что каждый пациент может проходить лечение только в одном санатории, и в одном санатории могут проходить лечение несколько пациентов. Поэтому степень связи ЛЕЧИТСЯ равна *n* : 1. Если допустить, что наша БД будет содержать сведения только о пациентах, прошедших курс лечения в том или ином санатории, а в каждом санатории хоть кто-нибудь находился на лечении, то отсюда следует, что класс принадлежности обеих сущностей в связи ЛЕЧИТСЯ является обязательным.

Следовательно, в соответствии с перечисленными выше правилами образования отношений для заданных условий необходимо создать четыре отношения:

ПАЦИЕНТ (#Пцнт, ФИО, Гржд, #Бол, #Снтр, …);

БОЛЕЗНЬ (#Бол, БНзв, Симп, …);

САНАТОРИЙ (#Снтр, СНзв, Грд, …);

ЛЕЧАТ (#Снтр, #Бол).

Предположим, что имели место факты, представленные диаграммой ER-экземпляров на рис. 5.54.

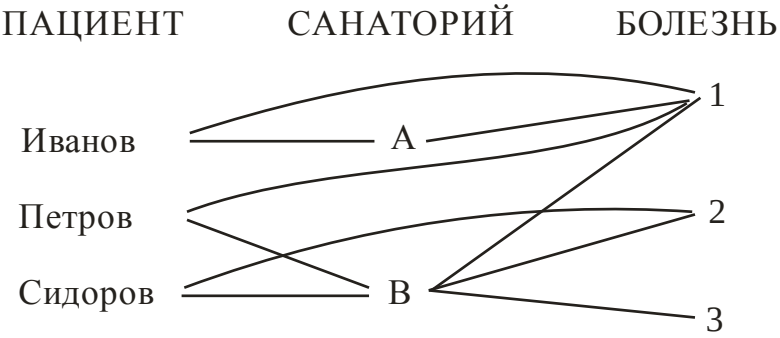


Рис. 5.54. Диаграмма ER-экземпляров

Отдельные экземпляры этих же отношений для наглядности представлены на рис. 5.55 в табличном виде.

ПАЦИЕНТ

#Пцнт	ФИО	Грожд	#Снтр	#Бол
1	Иванов	1960	a	1
2	Петров	1961	b	1
3	Сидоров	1971	b	2

САНАТОРИЙ

#Снтр	СНзв	Город
a	A	Омск
b	B	Томск
c	C	Барнаул

БОЛЕЗНЬ

#Бол	БНзв	...
1	Б1	
2	Б2	
3	Б3	

ЛЕЧАТ

#Снтр	#Бол
a	1
b	1
b	2
b	3

Рис. 5.55. Экземпляры отношений

Рассмотрим теперь более реальный пример и внесем изменения в связи БОЛЕЕТ и ЛЕЧАТ. Остальные условия задачи оставим неизменными.

С одной стороны, любой пациент может иметь несколько заболеваний, с другой стороны, одной и той же болезнью могут страдать разные люди. Поэтому связь между сущностями ПАЦИЕНТ и БОЛЕЗНЬ имеет степень, равную *m* : *n*. Будем считать, что в поликлинику не обращаются, как выражаются врачи, практически здоровые люди, поэтому в базе данных будут храниться сведения только о больных. Следовательно, сущность ПАЦИЕНТ в связи БОЛЕЕТ является обязательной. На территории, обслуживаемой поликлиникой, могут отсутствовать люди, страдающие какими-либо экзотическими или крайне редкими заболеваниями, но это не означает, что такие люди не могут появиться в будущем. Поэтому следует составить достаточно полный перечень заболеваний. Тогда сущность БОЛЕЗНЬ в этой же связи будем считать необязательной.

Предположим также, что один и тот же пациент (в разное время) может проходить лечение в разных санаториях, а в любом санатории лечится много людей. Следовательно, степень связи ЛЕЧИТСЯ равна *m* : *n*. Сущность ПАЦИЕНТ будем считать необязательной, поскольку кто-либо может не проходить лечения ни в одном санатории. Сущность САНАТОРИЙ отнесем к обязательным, так как трудно представить санаторий без пациентов.

Предположим далее, что среди прочих действительными являются следующие факты.

1. В санатории А лечат болезнь 1.
2. В санатории В лечат болезни 1, 2 и 3.
3. Иванов страдает заболеванием 1.
4. Иванов находился на лечении в санатории А.
5. У Петрова болезни 1 и 2.
6. Петров проходил лечение в санаториях А и В.

Мы можем уточнить эти утверждения следующим образом:

1) Иванов проходил лечение заболевания 1 в санатории А;

2) Петров проходил лечение заболевания 1 в санатории А, а лечение болезни 2 – в санатории В.

Перечисленные факты отражены в виде диаграммы ER-экземпляров на рис. 5.56.

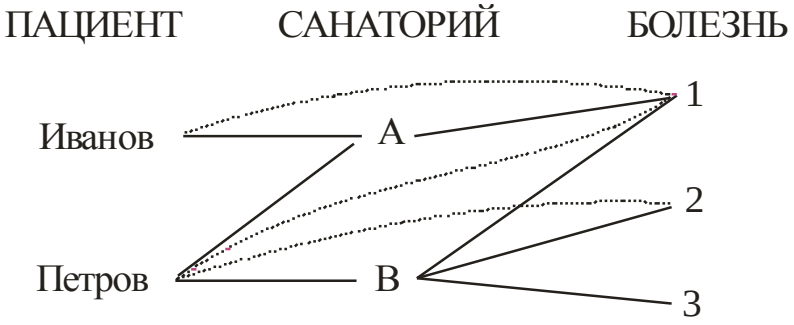


Рис. 5.56. Некорректная диаграмма ER-экземпляров

Использование бинарных связей в данном случае (см. рис. 5.56) не позволяет адекватно отобразить ситуацию. Если бы была создана соответствующая база данных, то не составляло бы труда определить, что Иванов находился в санатории А, где проходил курс лечения заболевания 1. Можно было бы вывести, что Петров находился в санатории А в связи с заболеванием 1, а в санатории В – по поводу болезни 2. Но также можно было бы придти к заключению о том, что Петров находился в санатории В по причине заболевания 1, что не соответствует действительности.

Трехсторонняя связь говорит больше (содержит больше информации), чем набор бинарных связей. Чтобы убедиться в этом, достаточно сравнить утверждение «Петров находился в санатории А в связи с заболеванием 1» и три следующих высказывания, содержащихся в бинарных связях:

1. Петров страдает заболеваниями 1 и 2.
2. В санатории В лечат болезни 1, 2 и 3.
3. Петров находился на лечении в санатории В.

Можно сделать вывод и о том, что Петров находился в санатории В в связи с заболеванием 1. Подобные ложные выводы называют *ловушкой при соединении отношений*.

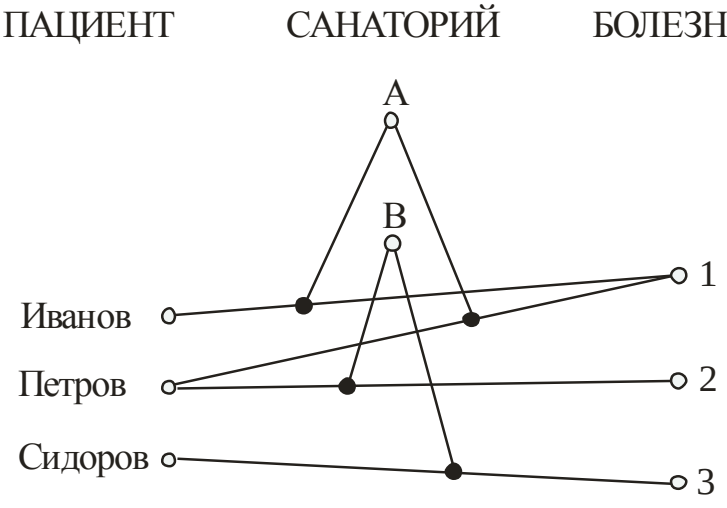


Рис. 5.57. Диаграмма ER-экземпляров

Неправильный вывод следует из того, что путь из вершины «Петров» в вершину 1 не является единственным (см. рис. 5.56). Решение подобных проблем заключается в переходе от бинарных связей к связям более высокого порядка.

Диаграммы ER-экземпляров, подобные рис. 5.56, в принципе не

позволяют правильно отображать трехсторонние связи. Для адекватного представления трехсторонних связей используют диаграммы ER-экземпляров, сходные с показанной на рис. 5.57. На нем сущности представлены светлыми кружками, а связи между экземплярами сущностей – черными кружками. Диаграмма ER-типа для отображения трехсторонних связей дана на рис. 5.58, который соответствует описанию ситуации с лечением больных в санаториях.

Создание предварительных отношений в случае трехсторонних связей осуществляется по следующему правилу. *В случае трехсторонней связи необходимо создать четыре отношения. Из них по одному отношению создается для каждой сущности и ключ каждой сущности должен использоваться в качестве первичного ключа соответствующего отношения. Четвертое отношение создается для связи и в число его атрибутов должны быть включены ключи каждой из трех сущностей.*

Пример корректной диаграммы ER-типов для трехсторонних связей приведен на рис. 5.58. Данная диаграмма и рис. 5.57 позволяют описать ситуацию однозначным образом, чего нельзя сказать о диаграмме на рис. 5.56.

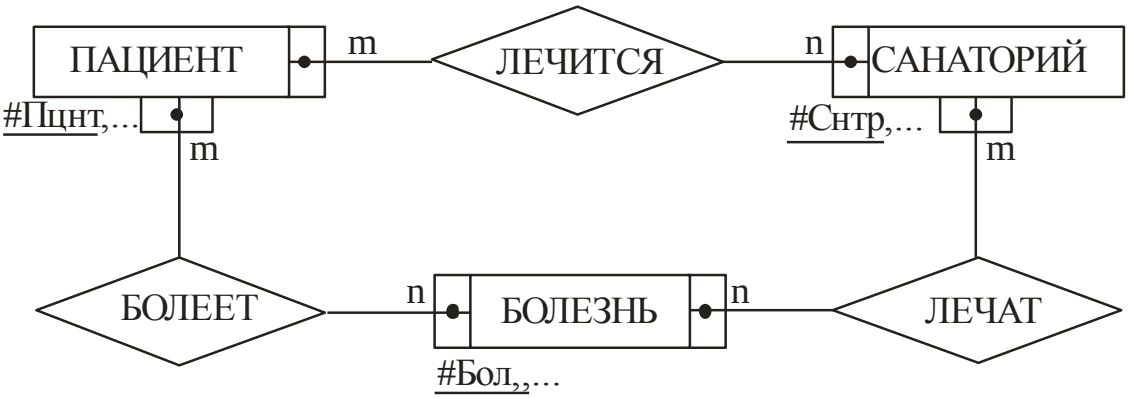


Рис. 5.58. Диаграмма ER-типа для трехсторонней связи

Сформулированное выше правило обобщается на случай *n*-сторонних связей очевидным образом: для представления каждой такой связи требуется $n + 1$ отношение.

5.18. Роли и их представление

До настоящего момента мы рассматривали связи между различными типами сущностей. Но таким же образом разнообразные связи могут существовать между различными экземплярами одной и той же сущности. Примером может служить связь РУКОВОДИТ, существующая между сотрудниками организации. Так, директор руководит своими заместителями, заместитель по производству руководит начальниками отделов, начальники отделов руководят руководителями групп, а последние – исполнителями.

Если брать пример из области геомоделирования, то связи между различными экземплярами одной сущности возникают при представлении отношения агрегации между объектами, когда одни экземпляры некоторой сущности (объекты) являются компонентами экземпляров той же сущности (объектов). Так, коробка здания, крыльцо, приямок и вход в подвал (объекты)

являются компонентами здания (также объекта). В данном случае более сложный в структурном отношении объект (здание) играет роль агрегата, а остальные объекты – его составных частей. В свою очередь, крыльцо также можно рассматривать как агрегат, состоящий из таких компонентов, как площадка крыльца и некоторое число ступеней.

В отношении агрегации можно выделить две сущности: АГРЕГАТ и КОМПОНЕНТ. Назовем связь между агрегатом и его компонентами СОДЕРЖИТ. Эта связь имеет степень 1 : *n*, и обе ее сущности являются обязательными. Диаграмма ER-типа связи между агрегатом и его компонентами представлена на рис. 5.59.

Рис. 5.59. Связь между агрегатом и компонентом

В соответствии с приведенным выше правилом в данном случае необходимо создать два предварительных отношения:

АГРЕГАТ (#Агр,…);

КОМПОНЕНТ (#Комп,…,#Агр).

Однако, при подобном представлении отношения агрегации возникает проблема избыточных данных. Нетрудно видеть, что некоторые неключевые атрибуты будут общими для обеих сущностей. Обратим внимание на то, что общими будут не значения атрибутов, а их наименования. И агрегат, и компонент будут иметь атрибуты Тип (код объекта в соответствии с принятой системой классификации и кодирования), Лок (тип локализации объекта), XYZ (положение объекта). Если эти атрибуты добавить в оба отношения, то они примут вид:

АГРЕГАТ (#Агр, Тип, Лок, XYZ, …);

КОМПОНЕНТ (#Комп, Тип, Лок, XYZ, …, #Агр).

Но, согласно общему правилу, каждый из неключевых атрибутов должен размещаться только в одном отношении! Поэтому мы не можем принять такое решение.

Можно попытаться искусственно разделить каждый такой неключевой атрибут на два атрибута. Так, атрибут Тип можно разбить на Атип (тип объекта для агрегата) и Ктип (тип объекта для компонета). Аналогичным образом из атрибута Лок можно создать атрибуты Алок и Клок и т. д. Но это также плохая идея. Если обратить внимание на объект «крыльцо», то можно заметить, что в одном случае он выступает в роли компонента, а в другом – в роли агрегата. При этом совершенно не ясно, считать ли крыльцо агрегатом или компонентом. Другие типы объектов могут вести себя так же.

Решение проблемы представления различных ролей состоит в следующем. Все объекты рассматриваются именно как сущности, а сущности АГРЕГАТ и КОМПОНЕНТ трактуются как *роли*, которые могут играть сущности. Соответствующая диаграмма ER-типов представлена на рис. 5.60.

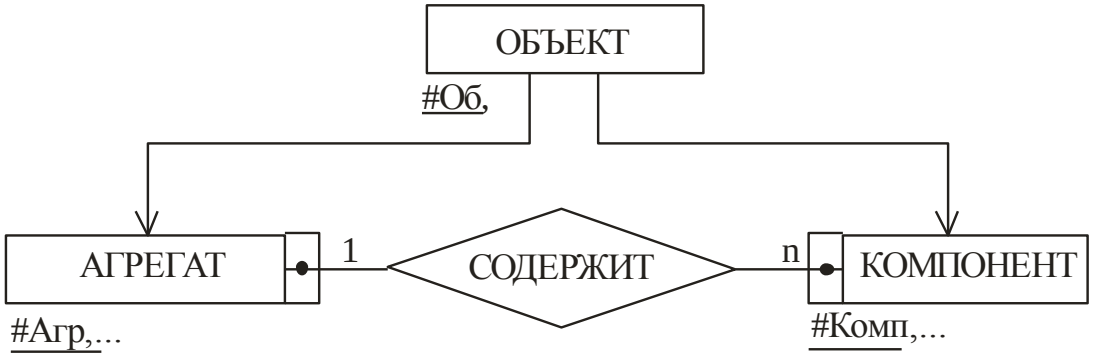


Рис. 5.60. Связь между агрегатом и компонентом

Для моделирования ролей используется следующее правило создания предварительных отношений. Исходная сущность используется для создания первого отношения, первичным ключом которого служит ключ сущности. Каждой связи между сущностями ставится в соответствие одно отношение, в котором каждая роль рассматривается как обычная сущность.

В соответствии с данным правилом, в нашем примере с объектами должны быть созданы три предварительных отношения:

ОБЪЕКТ (# Об, Тип, Лок, XYZ, …,);

АГРЕГАТ (#Агр, …);

КОМПОНЕНТ (#Комп, …, #Агр).

В отношении ОБЪЕКТ содержится информация, общая для любых объектов. В отношениях АГРЕГАТ и КОМПОНЕНТ должны быть помещены атрибуты, отражающие специфику ролей в данной связи. Тогда каждый атрибут будет располагаться на своем естественном месте. Таким дополнительным атрибутом, характеризующим роль КОМПОНЕНТ, мог бы быть, например, знаменатель масштаба M. Если при некотором значении масштаба картографическое изображение агрегата не строится, то естественно, что не строится также изображение его компонентов. Кроме того, значение M могло бы использоваться следующим образом: если значение знаменателя m текущего масштаба изображения на экране отвечает условию $m \le M$, то такой компонент выводится на экран, в противном случае – не выводится.

Связи, соединяющие две роли одной исходной сущности, называются рекурсивными. Это определение следует понимать в том смысле, что если роль сущности одного типа связана с ролью сущности другого типа, то такая связь уже не является рекурсивной.

В критических, с точки зрения скорости обработки, ситуациях в отношение, соответствующее исходной сущности, могут добавляться атрибуты, указывающие роль сущностей.

5.19. Другие нормальные формы

Кроме нормальной формы Бойса – Кодда известны и другие нормальные формы.

Это уже упоминавшаяся 1-я нормальная форма (1НФ): отношение *R* находится в *первой нормальной форме*, если оно не содержит повторяющихся кортежей. 1НФ является исходной для получения других нормальных форм,

получаемых последовательным наложением дополнительных ограничений на функциональные зависимости.

Отношение *R* находится во *второй нормальной форме* (2НФ), если оно находится в первой нормальной форме и каждый его атрибут, не являющийся основным, полностью (или функционально полно) зависит от любого возможного ключа отношения *R*. При разбиении исходного отношения, находящегося в первой нормальной форме, на отношения во второй нормальной форме используется принцип, в соответствии с которым атрибуты, функционально зависящие от одного основного атрибута, вместе с ним образуют одно новое отношение.

Вторая нормальная форма может произвести впечатление некоторой избыточности. Но если отношение находится только в 1НФ, то с ростом базы данных нельзя гарантировать ее целостность. Следовательно, возникает риск создания информационных моделей, неадекватных предметной области.

Отношение *R* находится в *третьей нормальной форме* (3НФ), если оно находится во второй нормальной форме, и каждый неключевой атрибут *R* нетранзитивно зависит от любого возможного ключа *R*. Это определение можно дать в «более интуитивной» форме: отношение *R* находится в третьей нормальной форме, если оно находится во второй нормальной форме, и все его неключевые атрибуты взаимно независимы.

Отношения, находящиеся в НФБК, одновременно находятся и в третьей нормальной форме. Но существуют нормальные формы, накладывающие более сильные ограничения, чем НБФК, и разработанные для избавления от тех нежелательных свойств, которыми могут обладать отношения, даже находящиеся в нормальной форме Бойса – Кодда. Такими нормальными формами являются четвертая (4НФ) и пятая (5НФ). Таким образом, НФБК занимает промежуточное положение между 3НФ и 4НФ.

Определение четвертой нормальной формы может быть дано с помощью понятия многозначной зависимости, являющейся обобщением функциональной зависимости (в том смысле, что любая функциональная зависимость является и многозначной, но не наоборот).

Пусть *X, Y* и *Z* – подмножества множества атрибутов отношения *R*. Говорят, что *Y многозначно зависит* от *X* тогда и только тогда, когда множество значений *Y*, соответствующее заданной паре значений (X, Z) зависит только от *X*, но не зависит от *Z*. Многозначная зависимость между атрибутами *X* и *Y* изображается с помощью двойной стрелки $X->>Y$, что читается как «*X* многозначно определяет *Y*» или как «*X* двойная стрелка *Y*».

Доказано, что для отношения *R*(*X*, *Y*, *Z*) многозначная зависимость $X->>Y$ выполняется тогда и только тогда, когда выполняется многозначная зависимость $X->>Z$. Следовательно, многозначные зависимости всегда образуют связанные пары. По этой причине они чаще всего представляются вместе, что в символьном виде записывается как $X->>Y \mid Z$.

В свете этого определения функциональную зависимость можно определить как многозначную зависимость, в которой множество зависимых

значений, соответствующее заданному детерминанту, всегда является одноэлементным множеством.

Отношение *R* находится в *четвертой нормальной форме* тогда и только тогда, когда существуют такие подмножества *X* и *Y* атрибутов отношения *R*, что имеет место нетривиальная многозначная зависимость $X->>Y$. Все атрибуты отношения *R* при этом также функционально зависят от атрибута *X*. *Тривиальной зависимостью* называется зависимость $X \to Y$, если *Y* является подмножеством *X*, то есть $Y \subseteq X$.

Переход от одной нормальной формы к другой основан на понятии декомпозиции без потерь. Пусть, например, имеется отношение $R(X, Y, Z)$. Его разложение на два отношения $R1(X, Y)$ и $R2(Y, Z)$ будет корректным, и запрос к отношениям $R1(X, Y)$ и $R2(Y, Z)$ даст тот же результат, что и запрос к отношению $R(X, Y, Z)$ лишь при определенных условиях. Декомпозицию, отвечающую таким условиям, называют *декомпозицией без потерь при естественном соединении* (объяснение операции естественного соединения дается дальше). Если естественное соединение $R1(X, Y)$ и $R2(Y, Z)$ приводит к большему числу кортежей, чем в отношении $R(X, Y, Z)$, то такую декомпозицию называют *декомпозицией с потерями*.

Отсутствие потерь при декомпозиции будет тогда и только тогда, когда от общего атрибута (в нашем случае – *Y*) зависит хотя бы один атрибут из двух оставшихся, то есть имеет место $Y \to X$ или $Y \to Z$. При этом атрибуты *X*, *Y* и *Z* могут быть как простыми, так и составными.

5.20. Оценка нормализации

Использование ER-метода проектирования баз данных соответствует распространенному методу «от общего к частному», когда проектировщик начинает с самых общих представлений о предметной области и постепенно их детализирует.

Основное преимущество ER-метода заключается в его эффективном применении для баз данных с большим числом атрибутов. Выполнение анализа функциональных зависимостей в самой первой фазе проектирования делает сам процесс проектирования неуправляемым. Но такой анализ обязательно должен выполняться на заключительной стадии проектирования с целью получения результирующих отношений.

Нормализация отношений позволяет значительно снизить число ошибок при ведении баз данных, но делает работу с базой данных более сложной, поскольку пользователям приходится делать запросы к большему числу файлов. Следовательно, перед проектировщиком БД всегда возникает проблема оптимизации числа отношений в базе данных. Однозначного решения этой проблемы не существует.

Известны прецеденты, когда даже в очень больших БД ограничивались использованием одного универсального отношения. Оправданием подобных решений часто служат требование высокой скорости обработки данных и простота формирования запросов к БД. Чтобы обеспечить целостность данных

в подобных ситуациях, используется специально разработанное программное обеспечение.

В большинстве же случаев стремятся к получению баз данных, в которых каждое отношение находится либо в НФБК, либо в четвертой нормальной форме. Сторонники этой точки зрения на проектирование БД отмечают, что большинство пользователей не обладают теоретической подготовкой в области баз данных и не имеют представлений о скрытых опасностях, возникающих при использовании ненормализованных отношений.

Повышение скорости обработки при использовании СУБД может осуществляться применением индексных файлов, в которых в качестве индекса используется атрибут, наиболее часто служащий ключом поиска. Кроме того, рекомендуется такой способ повышения эффективности обработки данных, как отказ от хранения значений тех атрибутов, которые могут быть вычислены по значениям других атрибутов.

5.21. Реляционная алгебра

Преимущество реляционных моделей данных перед другими моделями состоит в простоте представления данных, использовании строгих математических методов проектирования их структуры, а также в возможности относительно простого манипулирования данными. Для целей манипулирования содержимым реляционных БД было разработано два формальных аппарата: реляционная алгебра (или алгебра отношений) и реляционное исчисление (или исчисление отношений).

Реляционной алгеброй называют систему операций манипулирования отношениями, каждый оператор которой в качестве своего операнда или операндов имеет одно или несколько отношений, а его результатом является новое отношение, получаемое по определенным правилам. Таким образом, и аргументами операций, и результатами операций (функциями) в реляционной алгебре являются отношения. В качестве основных операций в алгебре отношений используются пять: проекция, объединение, разность, декартово произведение и селекция. Остальные могут быть получены как некоторые комбинации основных операций.

Проекция отношения $R(X, Y, ...Z)$ состоит в том, что из него удаляются некоторые компоненты, а оставшиеся могут быть переупорядочены. Пусть $R(D_1, D_2, ..., D_n)$ – *n*-местное отношение, и i_j – различные целые числа от 1 до n $(1 \le i_j \le n)$. Тогда проекцией $\pi_{i_1, i_2, ..., i_k}(R)$ отношения R на компоненты $i_1, i_2, ..., i_k$ будет являться отношение $\Pi(D_{i_1}, D_{i_2}, ..., D_{i_k})$.

R

D1	D2	D3
A	a	1
B	b	2
C	c	3

D3	D1
1	A
2	B
3	C

Рис. 5.61. Проекция отношения

Рассмотрим в качестве примера отношение $R(D_1, D_2, D_3)$, где D_k – имя столбца (рис. 5.61). На том же рисунке представлена проекция $P = \pi_{3,1}(R)$.

Если в полученном в результате проекции отношении имеются группы одинаковых кортежей, то в каждой такой группе оставляют только один кортеж, а остальные вычеркивают. В записи $\pi_{i_1, i_2, \ldots, i_k}(R)$ числа i_j обозначают порядковый номер компоненты, но таким же образом могут использоваться имена атрибутов. Ту же проекцию *P* отношения *R* (см. рис. 5.61) можно записать как $\pi_{D_3, D_1}(R)$.

Для формального определения других основных операций реляционной алгебры используем два отношения $R(A, B, C)$ и $S(D, E, F)$.

Объединением отношений *R* и *S*, обозначаемым как $R \bigcup S$, называют отношение, каждый кортеж которого принадлежит *R* или *S*. Участвующие в объединении отношения *R* и *S* должны:

- иметь одинаковую длину;
- быть определенными на одних и тех же доменах, иначе объединение не будет иметь смысла. Пример объединения отношений приведен на рис. 5.62.

R

D1	D2	D3
1	11	101
2	15	121
5	14	113

S

D4	D5	D6
1	12	109
2	15	121
7	11	111

R *S*

1	11	101
2	15	121
5	14	113
1	12	109
7	14	111

Рис. 5.62. Объединение отношений

Разностью отношений *R* и *S*, обозначаемой как $R - S$, называют отношение, каждый кортеж которого принадлежит *R*, но не принадлежит *S*. Здесь отношения *R* и *S* также должны иметь одинаковую длину. Пример разности отношений приведен на рис. 5.63.

R

D1	D2	D3
1	11	101
2	15	121
5	14	113

S

D4	D5	D6
1	12	109
2	15	121
7	11	111

R *S*

1	11	101
5	14	113

Рис. 5.63. Разность отношений

Пусть *R* и *S* – отношения длины k_1 и k_2 соответственно. Тогда *декартовым произведением отношений* *R* и *S*, обозначаемым через $R \times S$, называют множество кортежей длины $k_1 + k_2$, первые k_1 компонентов которых образуют кортежи, принадлежащие *R*, а последние k_2 – кортежи, принадлежащие *S*. Пример декартова произведения отношений приведен на рис. 5.64.

R

D1	D2	D3
1	11	101
2	15	121
5	14	113

S

D4	D5	D6
1	12	109
7	11	111

R S

1	11	101	1	12	109
1	11	101	7	11	111
2	15	121	1	12	109
2	15	121	7	11	111
5	14	113	1	12	109
5	14	113	7	11	111

Рис. 5.64. Декартово произведение отношений

Пусть *F* – некоторая формула, содержащая следующие элементы:

- операнды, являющиеся константами или номерами компонентов;
- арифметические операторы сравнения ($<, \leq, =, \neq, >, \geq$);
- логические операторы ($\neg, \wedge, \vee$).

Тогда *селекция* $\sigma_F(R)$ есть множество кортежей *t*, принадлежащих *R*, таких, что при подстановке *i*-го компонента *t* вместо любого вхождения номера *i* в формулу *F* для всех *i* она станет истинной. Так, $\sigma_{i=j}(R)$ представляет собой множество тех кортежей отношения *R*, *i*-й компонент которых равен *j*-му компоненту. Выражение $\sigma_{1>'1'}(R)$, например, определяет множество кортежей отношения *R*, первый компонент которых больше 1 (рис. 5.65). В этом выражении следует обратить внимание на апострофы, заключающие константу 1; без них константа была бы неотличимой от номера компонента. Таким образом, операция *селекции* означает выборку из отношения *R* кортежей, отвечающих условию *F*.

R

D1	D2	D3
1	11	101
2	15	121
5	14	113

$1\ '1'(R)$

D1	D2	D3
2	15	121
5	14	113

Рис. 5.65. Операция селекции

Наряду с перечисленными операциями алгебры отношений существуют другие, которые могут быть выражены через основные. К ним относятся пересечение, частное, тета-соединение и естественное соединение.

Пересечением отношений R и *S,* обозначаемым как $R \bigcap S$, называют отношение, состоящее из кортежей, принадлежащих одновременно *R* и *S*. Из данного определения следует, что отношения *R* и *S* должны иметь одинаковую арность (длину). Пересечение отношений *R* и *S* может быть представлено как $R \bigcap S = R - (R - S)$ или симметричной формулой $S \bigcap R = S - (S - R)$. Пример пересечения отношений приводится на рис. 5.66.

R

D1	D2	D3
1	11	101
2	15	121
5	14	113

S

D4	D5	D6
1	12	109
2	15	121
7	11	111

R S

2	15	121

Рис. 5.66. Пересечение отношений

Пусть даны отношения *R1* и *R2* с длинами k_1 и k_2 соответственно, причем $k_1 > k_2$. *Частным отношений R1* и *R2*, обозначаемым через $R1 \div R2$, называют множество кортежей *t* длины $k_1 - k_2$, таких, что для всех кортежей *u* длины k_2, принадлежащих *R2*, кортеж *tu* принадлежит *R1*. Пример частного отношений дан на рис. 5.67.

R 1

D 1	D 2	D 3
1	11	101
2	15	121
1	15	121

R 2

D4	D 5
11	101
15	121
2	15

R 1 *R* 2

1

Рис. 5.67. Частное отношений

Тета-соединение (θ-*соединение*) *отношений R* и *S* по столбцам *i* и *j*, обозначаемое через $R \underset{i\,\theta\,j}{\triangleright\!\triangleleft} S$, где θ – арифметический оператор сравнения (<, ≤, =, ≠, ≥, >), есть краткая запись выражения $\sigma_{i\theta(k_R + j)}(R \times S)$, где k_R – длина отношения *R*. Иными словами, тета-соединение отношений *R* и *S* представляет собой множество тех кортежей в декартовом произведении *R* и *S*, в которых *i*-я компонента *R* находится в отношении θ с *j*-й компонентой отношения *S*. Если отношение θ есть оператор «=», то эту операцию называют *эквисоединением*. Рис. 5.68 содержит пример тета-соединения отношений.

R

D1	D2	D3
1	11	101
2	12	121
5	9	113

S

D4	D5	D6
1	12	9
7	11	11

$_{2>6}(R \quad S)$

1	11	101	1	12	9
2	12	121	1	12	9
2	12	121	7	11	11

Рис. 5.68. Пример тета-соединения отношений

Еще одной операцией является *естественное соединение* $R \triangleright\!\triangleleft S$ двух отношений, которое применимо только тогда, когда столбцы в отношениях *R* и *S* имеют имена. Построение естественного соединения осуществляется следующим образом:

1) определяются все кортежи декартова произведения $R \times S$;

2) для каждого атрибута *A*, имеющегося в обоих отношениях, выбираются те кортежи из $R \times S$, у которых совпадают значения в столбцах *R.A* и *S.A*. (Здесь *R.A* и *S.A* являются именами атрибута *A* в отношениях *R* и *S* соответственно.);

3) в выбранных кортежах для каждого указанного атрибута A удаляется $S.A$.

Формально операция естественного соединения определяется следующим образом: $R \triangleright\triangleleft S = \pi_{i_1, i_2, ..., i_m}(\sigma_{R.A_1 = S.A_1 \wedge, ..., R.A_k = S.A_k}(R \times S))$, где $i_1, i_2, ..., i_m$ есть упорядоченный набор всех компонентов $R \times S$ за исключением компонент $S.A_1$, $S.A_2$, …, $S.A_k$. Пример естественного соединения отношений дан на рис. 5.69.

R

D1	D2	D3
a	c	d
b	c	d
c	c	e
d	a	b

S

D2	D3	D4
c	d	b
c	d	f
a	b	c

$R \triangleright\triangleleft S$

D1	D2	D3	D4
a	c	d	b
a	c	d	f
b	c	d	b
b	c	d	f
d	a	b	c

Рис. 5.69. Естественное соединение отношений

Результат перечисленных выше операций всегда является отношением. Данное свойство реляционных операций называют *замкнутостью*. Оно имеет важное значение в связи с тем, что результат операции является объектом того же рода, что и объекты, над которыми производилась операция. Следовательно, над результатом операции можно осуществлять другие операции. Это означает, что операции могут быть *вложенными*, то есть выражения могут содержать в качестве операндов не только имена отношений, но и более простые выражения.

Другой особенностью выражений реляционной алгебры является то, что операции применяются *ко всему множеству кортежей*, то есть к отношению в целом, а не к отдельному кортежу. Это отличительное свойство реляционных систем, в нереляционных системах за одно обращение к базе данных обычно выполняется обработка отдельных записей.

Все операторы реляционной алгебры можно разбить на две группы. В первую группу включаются операторы, соответствующие традиционным операциям над множествами: объединение, пересечение, вычитание и декартово произведение. Во вторую группу входят операторы, обозначающие специальные реляционные операции: выборку, проекцию, деление и соединение.

Результатом *выборки* служит отношение, содержащее все кортежи, отвечающие некоторому заданному условию.

Результатом *проекции* отношения R является отношение, содержащее все кортежи отношения R после исключения из него некоторых атрибутов.

Результатом *соединения* двух отношений является отношение, каждый кортеж которого представляет собой сочетание кортежа из одного отношения с кортежем из другого отношения при условии, что имеется общее значение для одного или нескольких общих атрибутов.

Пять операций (выборка, проекция, объединение, вычитание и произведение) называют *примитивными*, имея в виду, что ни одна из них не может быть выражена через другие операции реляционной алгебры.

5.22. Реляционное исчисление

Реляционное исчисление, построенное на использовании средств исчисления предикатов, дает возможность представлять отношения и операции над ними в аналитическом виде. Формулы исчисления отношений имеют вид

$\{t \mid \varphi(t)\}$,

где t – так называемые *переменные выборки*, имеющие две разновидности: переменная-кортеж и переменная на доменах. *Переменная-кортеж* обозначает кортеж определенной длины. Выражение $t^{(n)}$ говорит о том, что кортеж t имеет длину n.

Для переменных на доменах формулы имеют вид:

$\{t_1 t_2 ... t_n \mid \varphi(t_1, t_2, ..., t_n)\}$,

где n – длина кортежа t, а $t_1, t_2, ..., t_n$ – переменные на доменах.

Формула φ с переменными-кортежами строится из элементарных выражений (термов, атомов) и набора операторов. Термами являются следующие выражения:

1. $R(s)$, где R – имя отношения, а s – переменная-кортеж; данное выражение означает, что s является кортежем в отношении R;
2. $t[i]\,\theta\, s[j]$, где t и s обозначают переменные-кортежи, а θ – арифметический оператор сравнения (<, ≤, =, ≠, ≥, >). Выражения подобного вида следует понимать в том смысле, что i-я компонента кортежа t находится в отношении θ с j-й компонентой кортежа s. Например, запись $t[i] = s[1]$ означает, что i-я компонента кортежа t равна 1-й компоненте кортежа s;
3. $t[i]\,\theta\, c$ и $c\,\theta\, t[i]$, где c – некоторая константа. Первое выражение означает, что i-я компонента кортежа t находится в отношении θ с константой c, а второе указывает на то, что константа c находится в отношении θ с i-й компонентой кортежа t. Например, выражение $t[k] > 0$ есть запись того факта, что значение k-й компоненты кортежа t больше 0.

В реляционном исчислении, как и в исчислении предикатов, используются свободные и связанные переменные. Вхождение переменной t в формулу φ является *связанным*, если этой переменной предшествует либо квантор существования ∃, либо квантор общности ∀. Если такие кванторы отсутствуют, то переменную называют *свободной*.

Правильно построенными формулами в исчислении отношений с переменными-кортежами являются выражения следующих типов.

1. Каждый терм суть правильно построенная формула. Все вхождения переменных-кортежей в терм являются свободными переменными в такой формуле.

2. Если φ и ϕ суть правильно построенные формулы, то таковыми являются и выражения $\varphi \wedge \phi$, $\varphi \vee \phi$ и $\neg\varphi$. При этом выражение $\varphi \wedge \phi$ равносильно утверждению, что формулы φ и ϕ являются истинными; выражение $\varphi \vee \phi$ соответствует утверждению, что φ или ϕ являются истинными; выражение $\neg\varphi$ означает, что формула φ не истинна. Переменные-кортежи являются связанными или свободными в формулах $\varphi \wedge \phi$, $\varphi \vee \phi$ и $\neg\varphi$ таким же образом, как они являются свободными или связанными в формулах φ и ϕ. Вхождение переменной-кортежа t может быть свободным в формуле φ, но связанным в ϕ и наоборот.

3. Если φ – правильно построенная формула, то $(\exists t)(\varphi)$ и $(\forall t)(\varphi)$ – также правильно построенные формулы. При этом свободное вхождение переменной-кортежа t в формулу φ в формулах $(\exists t)(\varphi)$ и $(\forall t)(\varphi)$ становится связанным соответственно квантором существования и квантором общности.

Формула $(\exists t)(\varphi)$ означает, что существует хотя бы одно значение t, подстановка которого вместо всех свободных вхождений переменной t в формулу φ делает эту формулу истинной. Например, формула $(\exists t)(R(t))$ означает, что отношение R не пусто, то есть $R(t) \neq \varnothing$, или что существует хотя бы один кортеж $t \subseteq R$.

Формула $(\forall t)(\varphi)$ означает, что какой бы кортеж соответствующей длины ни подставить вместо всех свободных вхождений переменной t в формуле φ, она будет истинной.

Никакие другие выражения, кроме перечисленных выше, не являются формулами реляционного исчисления с переменными-кортежами. В формулах исчисления отношений принят следующий порядок старшинства: арифметические операции в обычном порядке, затем арифметические операции сравнения, кванторы $\exists$ и $\forall$, логические операторы $\neg$, $\wedge$ и $\vee$ в перечисленном порядке. При необходимости в формулах могут использоваться скобки.

Реляционное исчисление с переменными на доменах отличается от реляционного исчисления с переменными-кортежами тем, что в нем используются переменные доменов вместо переменных кортежей, то есть переменные, принимающие свои значения в пределах домена (а не отношения). Следует обратить внимание на то, что значениями переменных на доменах являются не сами домены, а их элементы.

Ниже даны определения *правильно построенных формул в реляционном исчислении с переменными на доменах*. Его основное отличие от исчисления отношений с переменными-кортежами состоит в том, что в нем отсутствуют переменные-кортежи и используются только переменные на доменах. *Переменная на домене* – это переменная, областью значений которой является соответствующий домен.

1. Если выражение является атомарным, то оно является правильно построенной формулой. Атомарные выражения могут иметь только два вида:

– $R(x_1\ x_2\ ...\ x_n)$, где R есть n-местное отношение, а каждое x_i ($i = 1, \ldots, n$) является константой или переменной на домене;

– $x\,\theta\,y$, где x или y суть константы или переменные на доменах, а θ есть арифметический оператор сравнения.

2. Если φ и ϕ суть правильно построенные формулы, то, как и в исчислении отношений с переменными-кортежами, таковыми являются и выражения $\varphi \wedge \phi$, $\varphi \vee \phi$ и $\neg\varphi$.

3. Выражения, содержащие кванторы $(\exists x)$ и $(\forall x)$, где x – переменная на домене, также являются правильно построенными формулами. Свободные и связанные переменные в реляционном исчислении с переменными на доменах определяются таким же образом, как и в исчислении с переменными-кортежами.

Выражения в исчислении отношений с переменными на доменах имеют вид

$$\{x_1 \ldots x_n \mid \varphi(x_1,\ldots,x_n)\},$$

где φ – формула, в которой только ее свободные переменные на доменах являются различными переменными $x_1, \ldots, x_n$.

Между реляционной алгеброй и реляционным исчислением существует связь, одни и те же операции над отношениями могут быть выражены с использованием языковых средств и того, и другого формализма.

Если R и S – два отношения, то их объединение $R \cup S$ на языке реляционного исчисления с переменными-кортежами может быть представлено в виде выражения

$$\{t \mid R(t) \vee S(t)\}.$$

Это выражение обозначает множество таких кортежей t, что t принадлежит R или S. Разумеется, что здесь имеет место все то же ограничение: оба отношения должны иметь одну и ту же арность.

Разность двух отношений $R - S$ может быть представлена в виде выражения

$$\{t \mid R(t) \wedge \neg S(t)\}.$$

Если R и S – два отношения, длина которых соответственно m и n, то их декартово произведение $R \times S$ можно представить как

$$\{t^{(m+n)} \mid (\exists u^{(m)})(\exists v^{(n)})\,(R(u) \wedge S(v) \wedge$$

$$\wedge\, t[1] = u[1] \wedge \ldots \wedge t[m] = u[m] \wedge t[m+1] = v[1] \wedge \ldots \wedge t[m+n] = v[n])\}.$$

Данное выражение обозначает множество кортежей t длины $m+n$, таких, что m его первых компонент образуют кортеж $u \subseteq R$, а остальные n компонент образуют кортеж $v \subseteq S$.

Проекцию $\pi_{i_1, i_2, \ldots, i_k}(R)$ отношения R можно записать как

$$\{t^{(k)} \mid (\exists u)\,(R(u) \wedge t[1] = u[i_1] \wedge \ldots \wedge t[k] = u[i_k])\}.$$

Наконец, селекция $\sigma_F(R)$ может быть представлена выражением

$$\{t \mid R(t) \wedge F'\},$$

где формула F' получена из F заменой каждого операнда i, обозначающего i-й компонент, на $t[i]$.

Существует также связь между исчислением отношений с переменными-кортежами и исчислением отношений с переменными на доменах.

Три рассмотренных математических аппарата (реляционная алгебра, реляционное исчисление с переменными-кортежами и реляционное исчисление с переменными на доменах) считаются эквивалентными по своим выразительным возможностям и служат основой языков манипулирования данными и языков запросов. Можно заметить, что реляционная алгебра основывается на теории множеств. В реляционных исчислениях кортежи отношений рассматриваются как истинные утверждения о некоторых фактах предметной области и используется математический аппарат исчисления предикатов.

5.23. Базы данных и ГИС

Изучение теоретических основ баз данных пользователями систем геомоделирования необходимо в связи с тем, что с появлением и развитием возможностей персональных компьютеров реляционные базы данных получили чрезвычайно широкое распространение. В своей практической деятельности самые разные специалисты вынуждены не только пользоваться уже готовыми базами данных, но иногда заниматься разработкой их структуры. Создание и поддержание в актуальном и целостном состоянии моделей предметной области (баз данных) является основной профессиональной обязанностью специалистов по практической геоинформатике.

В одной из статей сообщалось, что в результате исследования баз данных о личном составе ВМФ США было установлено, что сведения о более чем 50 % (!) служащих содержали ошибки. Содержание настоящей главы свидетельствует о том, что одной из потенциальных причин возникновения ошибок в БД является наличие ненормализованных отношений. Как мы надеемся, нам удалось показать, что разработка структуры баз данных не может основываться на интуитивных представлениях. Проектирование БД является проблемой более сложной, чем это может показаться на первый взгляд. Разработка корректной схемы базы данных возможна только при тщательном и последовательном анализе функциональных зависимостей между атрибутами отношений.

Необходимость в изучении математических основ проектирования баз данных возникает в связи с тем обстоятельством, что в геоинформационных и автоматизированных картографических системах для хранения семантической информации все чаще применяются коммерческие реляционные СУБД. Если на первых этапах для ГИС были характерны фиксированный набор моделируемых объектов и файловая организация данных, то сегодня последняя нередко используется только для хранения геометрических (пространственных) данных. Используемое в ГИС (возможно, не совсем корректно) понятие атрибутивной информации заимствовано из области обработки данных. Учитывая, что в системах геомоделирования может храниться колоссальный объем данных,

значимость этапа проектирования структуры геоинформационных моделей нельзя переоценить.

Проблема проектирования, разработки и использования баз данных является достаточно обширной областью знаний. В частности, она включает такие вопросы, как проектирование физической структуры данных, обеспечение конфиденциальности и безопасности данных, распределенные базы данных и другие. И эта область знаний интенсивно развивается. В ней появились такие новые направления, как объектно-ориентированные базы данных, семантизация баз данных и др. По подсчетам известного американского специалиста в области обработки данных К.Дж. Дейта, в мире ежегодно публикуется около 100 тысяч страниц по самым различным вопросам теории баз данных, и, по его же мнению, обработка данных является областью, в которой невозможно знать все. Его фундаментальный учебник объемом почти 800 страниц без претензий назван «Введением в системы баз данных».

В данной главе автор ставил перед собой цель показать необходимость изучения теории реляционных баз данных геодезистами, картографами и специалистами по практической геоинформатике, не имеющими какой-либо подготовки в этой области знаний. Топографо-геодезическое и картографическое производство по своей природе всегда было и будет информационным. В настоящее время происходит его постепенная трансформация в геоинформационное производство. Поэтому утверждения о необходимости знания упомянутыми специалистами теоретических основ обработки данных представляются естественными и не требующими доказательств в силу своей очевидности. По нашему мнению, основы теории баз данных должны изучаться в вузах студентами геодезической и, в особенности, картографической специальностей.

Автор надеется, что приведенный минимальный объем сведений из теории реляционных баз данных послужит стимулом для ее более глубокого изучения упомянутыми специалистами.

Особого внимания заслуживает начавшийся процесс семантизации баз данных, под которой понимается использование идей из области искусственного интеллекта в области обработки данных. Его значимость такова, что авторы [8, с. 587] не считают нужным сдерживать эмоции: ««Развитие систем баз данных первоначально было мотивировано потребностью в эффективных средствах хранения, манипуляции и извлечения большого количества разнообразных данных. По мере того, как в достижении этих целей наблюдался прогресс, возникла дополнительная заинтересованность в возможности задавать информационным системам правила, применяемые к хранимым фактам (данным), позволяющие выводить из них другие факты. Это волнующая перспектива, поскольку реализация таких средств подняла бы возможность обеспечения информационных потребностей на новый уровень».

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Дейт К.Дж. Введение в системы баз данных. 6-е изд. – Киев: Диалектика, 1998. – 784 с.

2. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ. – М.: Мир, 1991. – 252 с.

3. Калиниченко Л.А. Методы и средства интеграции неоднородных баз данных. – М.: Наука. Глав. ред. физ.-мат. лит., 1983. – 424 с.

4. Леонг-Хонг Б., Плагман Б. Системы словарей-справочников данных. – М.: Финансы и статистика, 1986. – 311 с.

5. Полищук Ю.М., Хон В.Б. Теория автоматизированных банков информации. – М.: Высш. шк., 1989. – 184 с.

6. Системы управления базами данных и знаний. Справ. изд. / Под ред. А.Н. Наумова. – М.: Финансы и статистика, 1991. – 352 с.

7. Тиори Т., Фрай Дж. Проектирование структур баз данных. В 2-х кн. – М.: Мир, 1985.

8. Хансен Г., Хансен Дж. Базы данных: разработка и управление. – М.: ЗАО «Издательство БИНОМ», 1999. – 704 с.

9. Codd E.F. A Relational Model of Data for Large Shared Data Banks //CACM – 1970. – 13. – № 6.

6. СИСТЕМЫ, ОСНОВАННЫЕ НА ЗНАНИЯХ

Традиционное применение ЭВМ связано с решением вычислительных (инженерных и экономических) задач, разработкой автоматизированных систем управления, информационно-поисковых систем и т. п. Характерной особенностью подобных задач является существование четких алгоритмов их решения или хотя бы принципиальная возможность алгоритмизации. Да и само назначение ЭВМ подчеркивается в их названии.

Алгоритмом называют последовательность действий, направленных на достижение заданной цели или решение поставленной задачи. Алгоритм отличается от других предписаний (текстов, побуждающих к действию) свойствами, вследствие которых существует возможность его исполнения автоматом и к которым относят: понятность, дискретность, точность, результативность и массовость.

Понятность алгоритма для конкретного исполнителя означает, что алгоритм содержит указания о выполнении только тех действий, и о проверке только тех свойств объектов, которые входят в систему команд исполнителя.

Дискретность алгоритма означает последовательное выполнение команд с точной фиксацией момента завершения выполнения одной команды и начала выполнения следующей.

Точность понимается как такое свойство алгоритма, когда после выполнения каждой команды известно, что алгоритм завершен, либо известна следующая выполняемая команда.

Результативность алгоритма состоит в том, что после конечного числа шагов исполнение алгоритма заканчивается решением задачи либо возникновением ситуации, когда исполнение алгоритма по некоторой причине не может быть продолжено.

Массовость означает применимость алгоритма ко всему множеству задач, для решения которых он разрабатывался.

Но существует весьма широкий класс задач, для решения которых, по крайней мере – в настоящее время, точные методы не могут быть разработаны. Кроме того, многие задачи имеют не количественный, а качественный характер. Примерами таких задач являются головоломки и интеллектуальные игры разной сложности (от «крестиков и ноликов» до шахмат). Качественные задачи существуют не только в области развлечений, но и в производственной сфере, и некоторые авторы считают, что они более распространены, чем количественные задачи.

Методологию решения трудноформализуемых задач относят к области, получившей обобщенное название искусственного интеллекта (ИИ). В настоящее время во многих слабо формализованных прикладных областях, например, таких, как медицина, геология, археология, юриспруденция и т. п., широко и успешно применяются системы обработки, называемые интеллектуальными. Их отличительной особенностью является то, что они воспроизводят логику специалистов, решающих задачи в конкретной проблемной области, в связи с чем их обычно называют *экспертными*

системами. Учитывая специфику информации о географическом пространстве и процессов ее переработки, следует признать наиболее перспективной и даже необходимой разработку автоматизированных картографических систем и ГИС как интеллектуальных систем.

6.1. Определение искусственного интеллекта

При разработке искусственного интеллекта перед разработчиками прежде всего возникает вопрос его определения. Затруднения начинаются уже с дефиниции исходного понятия – интеллекта. Практически все определения искусственного интеллекта отталкиваются от интеллекта «естественного», и таким образом возникает проблема отношения искусственного и естественного интеллектов.

В толковых словарях интеллект определяется как мыслительная деятельность, мышление, рассудок, разум, ум, сознание и противопоставляется чувствам, воле и интуиции. В [17, с. 22] дано следующее определение: «Мышление – активный процесс отражения объективного мира в понятиях, суждениях, теориях и т. п., связанный с решением тех или иных задач, высший продукт особым образом организованной материи – мозга».

Специалистами по информатике подобные определения отвергаются как нестрогие и вводятся так называемые «рабочие» определения. Одно из таких определений дано в [15], где интеллектом называют способность организма или устройства добиваться измеримой степени успеха при поиске одной из множества целей при многообразии внешних условий. Таким образом, интеллект связывается с *целеполаганием*, со способностью ставить цели.

В [15] проблема соотношения искусственного и естественного интеллектов рассматривается на трех уровнях. Первый уровень – оценка принципиальной возможности или потенциальной осуществимости искусственного интеллекта. Вопрос о его потенциальной осуществимости решается на основе известных на текущий момент фундаментальных законов природы. Ни один из известных сегодня таких законов не может служить доказательством невозможности создания искусственного интеллекта. При этом, правда, остается открытым вопрос, не будет ли обнаружен такой закон в будущем.

Однако потенциальная осуществимость некоторой идеи еще не означает возможность ее технической реализации. В конце концов, потенциальная возможность создания ЭВМ существовала и во времена средневековья. Поэтому на втором уровне выполняется оценка возможности реализации искусственного интеллекта при современном технологическом уровне. Ответ на этот вопрос также положительный. Наконец, на третьем уровне рассматривается вопрос об экономической целесообразности.

С функциональной точки зрения «мышление представляет собой процесс отражения действительности, в ходе которого человек … разрабатывает планы и в конечном счете решает те или иные задачи» [20, с. 13].

Далее со ссылкой на Д.А. Поспелова и В.Н. Пушкина отмечается, что «…любое продвижение вперед в области обучения машин … связано с разработкой процедур *машинного формирования понятий,* с проблемой

автоматического абстрагирования и обобщения», и делается вывод: «Представляется разумным, что прежде всего нужны такие теории, которые трактовали бы эту проблему так, чтобы получаемые результаты могли быть интерпретированы как в логико-психологических (человечески-ориентированная эпистемология), так и в автоматизационных (эпистемология машинная) терминах» [20, с. 294].

По Н. Нильсену [3], искусственный интеллект – это инженерная дисциплина, целью которой является создание конструкций, которые могут считаться «разумными» системами.

Уже после первых попыток применения ЭВМ для решения интеллектуальных задач (конец 1950-х – начало 1960-х гг.) было осознано, что аппарата только формальной логики недостаточно, и что разработка технических систем, выполняющих интеллектуальные функции, включая абстрактное мышление, должна основываться на результатах наук, исследующих различные аспекты мышления. Американскими специалистами в области искусственного интеллекта Дж. Маккарти и Р.Дж. Хейесом было высказано мнение, со временем ставшее общим, что разработчики систем искусственного интеллекта должны учитывать результаты исследований мышления, полученные философами. С другой стороны, попытки создания искусственных интеллектуальных систем побудили некоторых философов к анализу особенностей мышления, благоприятствующих или препятствующих выполнению интеллектуальных функций техническими системами.

В [15, с. 323] машинным интеллектом называют «следующие свойства машины:

а) богатство хранящегося в ней запаса «знаний» и «умений» (констант, понятий, всевозможных стандартных алгоритмов), удобство использования этого запаса, а также его пополнения путем обучения в процессе эксплуатации (что является особенно интересным);

б) логическое совершенство методов переработки информации в машине, приема и выдачи информации, а также способность машины к организации сложного вычислительного процесса в целом;

в) понимание машиной входных языков высокого уровня. Степень развития этих свойств и определяет уровень «машинного интеллекта».

Ясно, что данные свойства в совокупности имитируют такие черты человеческого интеллекта, как эрудиция, сообразительность, продуктивность и организованность, восприимчивость к входной информации (в частности, понятливость). Хотя человеческий интеллект этими чертами далеко не исчерпывается, именно приведенные свойства наиболее существенны с точки зрения их использования при решении задачи автоматизации творческих процессов».

Можно отметить роль организации знаний, хотя авторы этого не подчеркнули. Организованность и способность к самоорганизации являются важнейшими чертами человеческого интеллекта. Отсюда следует необходимость структуризации знаний в интеллектуальных системах.

Наиболее полный перечень требований к искусственному интеллекту приводится в [20]. Интеллектуальная техническая система должна:

1) иметь внутреннюю модель внешнего мира, что позволяет ей иметь относительную самостоятельность;

2) обладать способностью к пополнению знаний о внешнем мире;

3) уметь выполнять рассуждения о проблемной области, извлекать информацию, представленную в описании проблемной области имплицитно (неявно);

4) выполнять свои функции в условиях неопределенности или нечеткости;

5) быть способной к диалогу, при необходимости уметь объяснить ход рассуждений;

6) обладать способностью к адаптации.

Специалисты в области инженерии знаний допускают, что в конкретной системе искусственного интеллекта, в зависимости от ее специфики, некоторые из указанных требований могут не выполняться, но наличие внутренней модели мира является обязательным условием.

Таким образом, под искусственным интеллектом обычно понимают технические системы, целью которых является решение таких задач, с которыми ранее мог справиться только человек. В этом определении необходимо подчеркнуть значение факта получения результата решения. Важно, что результат должен быть аналогичным тому, что был бы получен человеком на основе размышлений.

В связи с такой формулировкой задачи разработки искусственного интеллекта требует объяснения вопрос, в каком смысле результат, полученный технической системой, совпадает с результатом, полученным человеком.

Согласно [20], ответ на поставленный вопрос заключается в следующем. Итогом мыслительного процесса является решение некоторой задачи. В сознании человека, решившего задачу, это решение является совокупностью субъективных образов (суждений, идей и т. п.), но оно может быть представлено в объективной форме – в виде сообщения с помощью какой-либо знаковой системы.

Любой другой человек, знакомый со знаковой системой, может узнать результат решения после получения сообщения. В результате интерпретации сообщения в сознании его получателя индуцируются субъективные образы, близкие к образам отправителя сообщения. При этом для получателя совершенно безразлично, были ли субъективные образы в сознании отправителя сообщения, но представляет интерес *содержание* сообщения, представленного в знаковой форме.

Если сообщение в виде последовательности знаков, сформированное машиной, вызывает у его получателя те же образы, что и сообщение, отправленное человеком, то тогда для получателя не существенно, кто в действительности являлся отправителем сообщения, и с его (получателя) точки зрения решение, полученное технической системой, совпадает с результатом, полученным человеком.

Таким образом, при решении задач мыслительная деятельность человека заключается в манипуляции определенными образами и в их описании с помощью знаковой системы. В отличие от человека, техническая система не создает субъективных образов проблемной области, но такая задача перед ней и не ставится. Именно эту неспособность машин к генерации образов имеют в виду, когда говорят, что машины не умеют мыслить. Если согласиться с таким утверждением, то нам пришлось бы признать, что мышлением можно назвать только человеческое мышление, поскольку вполне возможно, что манипуляции образами могут оказаться особенностью мыслительной деятельности, присущей только человеку.

В итоге мы приходим к выводу, что с прагматической точки зрения результаты функционирования технической системы и мыслительной деятельности человека совпадают, если идентично содержание полученных знаковых последовательностей. Более того, если получателем сообщения является другая техническая система, то для нее также безразлично, кем была сформирована последовательность знаков. Перед ней также не ставится задача создания образов, а требуется выполнение определенных действий.

6.2. Искусственный интеллект как предмет исследований

Искусственный интеллект является обширной областью исследований, в которой выделяют четыре наиболее общих направления: бионическое, эвристическое, эволюционное и лингвистическое, называемое также семиотическим, знаковым или информационным [15].

В *бионическом* направлении непосредственными объектами исследования и моделирования являются структуры и процессы в нервной системе человека или животных. В основе бионического направления лежит рассуждение, что поскольку единственным объектом, способным мыслить, является человеческий мозг, постольку любое разумное устройство должно в той или иной мере воспроизводить его структуру. По аналогии с мозгом, состоящим из взаимодействующих клеток – нейронов, искусственно создаваемые структуры получили название *нейронных сетей*, или *нейросетей*.

Первая в мире нейросеть, предназначенная для моделирования процессов восприятия зрительных образов и названная PERCEPTRON, была создана в 1957 г. под руководством Ф. Розенблата. Исследования в данном направлении не получили в дальнейшем широкого развития по двум причинам: во-первых, результаты практических экспериментов оказались значительно хуже ожидаемых; во-вторых, Минским и Пейпертом было теоретически доказано, что подобные персептрону системы способны распознавать лишь сравнительно узкий класс образов.

Критики бионического направления отмечали, что подобие структур не является обязательным, и что определяющим фактором служит не конструктивное исполнение системы, а выполняемые функции. Имитация структуры «естественного» интеллекта является лишь одним из возможных путей создания интеллекта искусственного. Эта точки зрения подтверждается существованием технических устройств, не имеющих аналогов в природе.

Хотя со временем представления относительно возможностей нейросетей приобретали все более пессимистический оттенок, связанные с ними надежды не исчезли совсем. В 1980-х гг. в рамках правительственного проекта «ЭВМ пятого поколения» в Японии был создан первый в мире нейрокомпьютер. В настоящее время нейрокомпьютеры находят основное применение в области распознавания образов, в том числе – при обработке аэро- и космоснимков, что представляет интерес для систем геомоделирования.

В *эвристическом* направлении моделируется внешняя деятельность интеллекта, решающего задачи с элементами новизны. При этом предполагается, что мышление человека основано на некотором числе таких простых операций, как поиск, сравнение, модификация символов и т. п. В принципе решение новой задачи может осуществляться методом проб и ошибок. Тогда поиск решения задачи сводится к последовательному или случайному перебору в пространстве решений и может выполняться машиной. Но предварительно задача должна быть представлена в форме, допускающей ее решение методом перебора.

В силу очень большой размерности пространства решений метод проб и ошибок в чистом виде, то есть метод полного исчерпания, практически не применим без использования каких-либо стратегий направленного поиска, резко ограничивающих число просматриваемых промежуточных вариантов. Определение «эвристическое» в названии направления указывает на использование таких стратегий.

В процессе решения каждая задача разбивается на подзадачи, которые могут быть решены. Если какая-то подзадача не может быть решена, то она аналогичным образом разбивается на подзадачи и т. д. Такой подход первоначально основывался на «поиске в глубину», что влекло за собой необходимость анализа множества тупиковых ветвей. Позднее были разработаны более эффективные методы «поиска в ширину».

Первой эвристической программой явилась «Логик-теоретик», разработанная А. Ньюэлом и Г. Саймоном в середине 1950-х гг., но большее впечатление в свое время произвела программа GPS (General Problem Solver – универсальный решатель задач), разработанная ими же в 1960-х гг. Эта программа была «универсальной» в том смысле, что не требовала указания, к какой области относится решаемая задача. Но задача предварительно должна была представляться в терминах объектов проблемной области и применимых к ним операций.

Как отмечается в [20], универсальность такого рода возможна лишь в ограниченном мире математических головоломок с небольшим множеством состояний и с четкими формальными правилами. Решать реальные (практические) задачи указанные программы были не в состоянии.

Эволюционное направление, развившееся из теории самовоспроизводящихся автоматов Дж. фон Неймана, занимается изучением поведения и эволюции систем под влиянием внешней среды. В центре внимания этого направления является не то, что есть, а то, что может быть, если процессы будут развиваться в нужном направлении. Первые эксперименты по

моделированию процессов выживания и развития систем были осуществлены Л. Фогелем в 1966 г.

Роль автоматов (эволюционирующих организмов) в этих экспериментах играли алгоритмы, предсказывавшие последующие символы по заданным наборам символов. Символы интерпретировались как описание внешней среды, в которую погружен развивающийся организм, в прошлом, настоящем и будущем. Эволюция автомата трактовалась как его самоусовершенствование за счет мутаций – изменений алгоритма предсказаний.

С помощью датчика случайных чисел изменялись параметры формулы предсказания, и измененная формула включалась в новый алгоритм, который рассматривался как потомок старого алгоритма, то есть мутант. Если качество предсказания символов мутантом ухудшалось, то он «не выживал» и отбрасывался. Если же точность предсказания мутантом повышалась, то он становился объектом дальнейшей эволюции и порождал новых мутантов и т. д. Таким образом, прогрессивная эволюция генерировала все более добротные автоматы, решавшие задачи поведения в окружающей среде. Процесс эволюции заканчивался при достижении хорошего качества предсказания или когда исчерпывалось предельное число поколений.

Очевидно, что такое «эволюционное моделирование» очень сильно примитивизирует реальные эволюционные процессы и даже отдаленно не отражает ни сложность живых организмов, ни их эволюцию.

Наконец, *лингвистическое* направление изучает методы и средства представления внешнего мира в системах искусственного интеллекта. Исследования психологов показали, что в ходе решения задач человек воссоздает проблемную ситуацию в виде системы отношений между составляющими элементами, которая далее используется для построения моделей различного уровня. Относительно лингвистического направления в [15, с. 293] отмечалось: «В последнее время становится ясно, что такое моделирование внешней среды есть важнейшее, ключевое звено комплекса работ по автоматизации познавательных процессов». Лингвистический подход к моделированию внешнего мира основывается на том, что информация о моделируемых процессах представляется в форме их *языковой модели*.

Там же приводится точка зрения Дж. Маккарти, согласно которой проблема искусственного интеллекта сводится к решению двух проблем: проблемы эвристик и проблемы представления. Последнюю проблему Маккарти называл эпистемологической и понимал ее как *разработку методов машинного представления внешнего мира*. Содержание такого представления составляют данные и знания о внешнем мире.

6.3. Данные и знания

В обычном смысле *знания* трактуются как результат познания действительности, представляющий целостную систему абстрактных объектов, доступную для понимания человеком. В системах технической имитации интеллекта под знаниями понимается формализованная информация о конкретной выделенной области реального мира, используемая в процессе

логического вывода и позволяющая получать знания, представленные в системе в неявном виде.

Далее при описании задач будут использоваться понятия предметной области, проблемной области, данных и знаний. *Предметной областью* называют некоторую выделенную часть реального или мыслимого мира, для моделирования которой и создается информационная система. *Проблемная область* включает в себя предметную область и решаемые в ней задачи. Этот термин используется в тех случаях, когда хотят подчеркнуть, что интерес представляет не только моделируемая область, но и задачи, существующие в области моделирования. *Данными* называют входные, промежуточные и выходные сведения, используемые или получаемые в процессе решения конкретной задачи, то есть сведения, существующие только во время ее решения. В отличие от данных, *знания* используются при решении множества задач и хранятся в системе вне зависимости от того, решается в данный момент какая-либо задача или нет.

Одни и те же сведения могут быть и данными, и знаниями. Их отнесение к тому или иному классу зависит от способа использования информации в системе. Чтобы пояснить разницу между данными и знаниями, рассмотрим следующий пример. Пусть имеется программа для решения задач сфероидической геодезии. Но для решения любой задачи на эллипсоиде требуется знать его параметры, например, значения большой полуоси и эксцентриситета. Если эти значения хранятся в тексте программы в виде констант, то их ввод не требуется, что упрощает использование программы. Однако область ее применения является узкой, поскольку сводится только к одному эллипсоиду. В данном случае параметры земного эллипсоида представлены как знания.

Если в другой программе параметры эллипсоида должны вводиться при решении каждой задачи, то такая программа является универсальной в том смысле, что позволяет решать задачи на любом эллипсоиде. Но ее использование менее удобно, чем использование первой программы, так как требуются дополнительные затраты на ввод параметров эллипсоида. Параметры эллипсоида при этом представляют собой данные.

Очевидно, что наилучшим решением была бы третья программа, обеспечивающая ввод и хранение параметров каждого нового эллипсоида. Такая программа сочетала бы лучшие свойства двух первых программ. Параметры эллипсоида в ней выступают в роли знаний. В работе такой программы можно выделить два режима: режим обучения, когда осуществляется пополнение хранящихся знаний (ввод параметров нового земного эллипсоида), и режим решения задач. Уже этот небольшой пример демонстрирует потенциальные преимущества использования знаний.

Наиболее впечатляющим примером значения знаний является феномен использования любых программ, решающих, например, вычислительные задачи. Пользователь может не знать алгоритмов решения этих задач вообще. Все что от него требуется – это правильно ввести данные. По этой причине программное обеспечение было кем-то метко названо консервированными

знаниями. Таким образом, использование знаний позволяет хотя бы в некоторых случаях снизить требования к квалификации пользователей.

Перечисленные различия между данными и знаниями являются внешними. Наряду с общими чертами существуют глубокие и принципиальные различия между данными и знаниями. К ним относят внутреннюю интерпретируемость, структурированность, связность, семантическую метрику и активность знаний.

Внутренняя интерпретируемость. Каждый элемент знаний имеет уникальное имя, используемое для его поиска и манипуляции им. Если данные в памяти ЭВМ лишены имен, то в общем случае отсутствует возможность их идентификации. Они могут быть идентифицированы только соответствующей программой, извлекающей их в определенной точке алгоритма по указанию программиста. В отрыве от программы данные не имеют какого-либо смысла. Если, например, программа обрабатывает пространственные прямоугольные координаты, то только автор программы обладает знаниями о структуре данных, знает, в каком порядке расположены координаты, хранятся ли они по столбцам или построчно, и в каком порядке перечисляются координаты *x, y* и *z*. Автор программы может, конечно, описать структуру данных и сделать ее, таким образом, доступной для других программистов, но структура данных по-прежнему будет неизвестна машине.

Как было показано в предыдущей главе, для решения этой проблемы были разработаны СУБД. По существу, содержание схемы базы данных представляет собой знания о структуре всей базы данных, которая является отражением предметной области. Вследствие этого СУБД обладают возможностью внутренней интерпретации любых данных, хранимых в БД. Таким образом, появление концепции баз данных – это логичный шаг к системам, основанным на знаниях. В отличие от данных, знания могут быть всегда проинтерпретированы.

Структурированность. Чтобы информацией было удобно манипулировать, она должна обладать гибкой структурой. Это означает, что информация должна быть организована по «принципу матрешки», под которым понимают рекурсивную вложенность одних информационных структур в другие. Подобный способ организации информации подразумевает возможность включения любой информационной структуры в состав другой и извлечения из каждой информационной структуры ее любых составляющих.

Структуризация знаний может осуществляться с использованием классифицирующих отношений (отношений таксономии). Убедительным примером эффективности использования знаний является ситуация, когда наряду с индивидуальными свойствами элементов некоторого множества требуется знать их общие свойства, то есть свойства всего множества. При обработке данных общие свойства множества дублируются для каждого элемента множества. В системах, основанных на обработке знаний, для каждого элемента множества достаточно указать его индивидуальные свойства и принадлежность определенному множеству, что позволяет более эффективно использовать память ЭВМ.

Связность. Собственно говоря, структурированность информации уже представляет собой ее связность, но это только один вид связей. Требование связности следует понимать как возможность определения отношений любого вида между информационными структурами. В общем случае все множество возможных отношений принято делить на отношения структуризации, функциональные отношения, каузальные отношения и семантические отношения. Назначение отношения структуризации следует из его названия и состоит в описании информационной структуры. Функциональные отношения определяют отношения типа «аргумент – функция» и связаны с вычислением тех или иных функций. Их относят к процедурным знаниям. Каузальные отношения описывают причинно-следственные связи (типа «причина – следствие»). Все остальные отношения иногда объединяют под общим названием семантических отношений.

Связи подобного типа между понятиями, называемые ситуативными, дают возможность анализа знаний на целостность и непротиворечивость, а также проверки соответствия содержания базы данных ограничениям и условиям, содержащимся в базе знаний, что повышает достоверность баз данных.

Таким образом, системы управления базами знаний (СУБЗ) предоставляют пользователям более мощные средства управления моделью предметной области и, как следствие, – новые возможности. Пользователь системы, основанной на знаниях, может не только использовать структуры информации, реализованные в системе, но и создавать новые по мере необходимости. Следовательно, интеллектуальные системы характеризуются более высокой степенью адаптивности, чем системы обработки данных. В результате сокращается временной промежуток между моментом возникновения необходимости в решении новой задачи и моментом получения результатов решения.

Базы знаний являются не альтернативой базам данных, а их дальнейшим развитием. Базы знаний и базы данных представляют собой разные уровни представления информации.

Семантическая метрика. В некоторых приложениях требуется описывать степень близости различных информационных единиц. С этой целью вводится понятие семантического расстояния, характеризующего силу ассоциативной связи между двумя информационными единицами. Поясним это на следующем примере. Понятия «снег» и «белый» являются ассоциативно связанными. Равным образом, связаны понятия «конституция» и «демократия». Но этого нельзя сказать о паре «снег» и «конституция» или о паре «снег» и «демократия». Очевидно, что семантическое расстояние между «снегом» и «белым» меньше семантического расстояния между «снегом» и «конституцией».

Степень семантической близости между информационными единицами иногда называют отношением релевантности («релевантный» означает то же, что и «относящийся к делу» или «уместный»). Отношение релевантности позволяет отыскивать знания, близкие к уже найденным, когда выполняется

анализ некоторых типичных ситуаций («утро рабочего дня», «подготовка к зиме», «празднование дня рождения» и многие другие).

Активность. При традиционном разделении содержимого машинной памяти на данные и команды данные рассматриваются как пассивный элемент, а команды – как активный. Все происходящие в ЭВМ процессы представляют собой последовательное выполнение команд, по мере необходимости использующих те или иные данные. Для интеллектуальных систем такая ситуация считается неприемлемой. Существует мнение, что инициирование каких-либо процессов в машине должно определяться состоянием информационной базы, и причиной активизации интеллектуальной системы должны быть изменения, происходящие в информационной базе.

Считается также, что перечисленные особенности информационных единиц устанавливают различия между данными и знаниями. В настоящее время не существует интеллектуальных систем, в которых особенности знаний были бы реализованы полностью. В реальной действительности встречаются системы, в которых знания используются весьма слабо, и системы с развитыми интеллектуальными возможностями. Однако общие тенденции таковы, что постепенно системы обработки данных приобретают все больше черт, присущих системам, основанным на обработке знаний.

По своему содержанию знания подразделяются на декларативные (неалгоритмические), операционные (алгоритмические) и метазнания. По форме изложения декларативные и операционные знания могут быть представлены как в декларативной, так и в процедурной форме. Различия между декларативной и процедурной формой представления знаний соответствуют разнице между «знать что» и «знать как». Декларативное представление знаний не содержит в явном виде процедур, которые должны выполняться. Как правило, декларативные знания представляют собой некоторое множество утверждений, не зависящих от способа (когда и как) их использования. Описание предметной области (модель) в такой форме является, по существу, синтаксическим. Решение задач в таких системах основывается преимущественно на процедурах поиска в пространстве состояний. Реализация соответствующих процедур должна учитывать особенности моделируемой области, то есть семантику существующих в ней отношений. Таким образом, при декларативном представлении происходит разделение синтаксических и семантических знаний. Данное свойство позволяет характеризовать декларативные знания как более общую и универсальную форму представления знаний.

Процедурные знания представляют собой явные описания процедур как последовательности определенных действий. Их использование позволяет отказаться от хранения всех существующих в базе знаний отношений и хранить только определенную минимальную модель – описание предметной области в «свернутом» виде. Все необходимые состояния предметной области с помощью процедур могут быть сгенерированы из исходного. Представление знаний в форме процедур обеспечивает более быстрый поиск решений по сравнению с декларативным представлением. Однако декларативная форма представления

знаний превосходит процедурную по возможности накопления, модификации и развития знаний. Таким образом, оба способа представления знаний не исключают, а взаимно дополняют друг друга.

Знания обычно представляются в виде общих понятий, характерных для предметной области, то есть классов объектов и взаимосвязей между ними, процедур и правил манипулирования понятиями и конкретными фактами, а также информации о том, когда и как следует применять правила и процедуры [20].

Достоинство любой классификации заключается в том, что существенным образом снижаются требования к необходимому объему машинной памяти, так как, во-первых, достаточно хранить характеристики класса, а не каждого индивидного объекта, и, во-вторых, для каждого класса нижнего уровня иерархии требуется запоминание только его уникальных свойств, а остальные в любой момент времени могут быть заимствованы из классов верхних иерархических уровней [20].

Декларативные знания состоят из абстрактных объектов, называемых понятиями. Каждое понятие имеет одно или несколько *имен* (синонимов), *определение*, ту или иную *структуру* и входит в некоторую *систему понятий*. К декларативным знаниям относят также знания о связях между понятиями, о свойствах понятий и связях между свойствами понятий. Декларативные знания подразделяют на *концептуальные* (или *интенсиональные*, *понятийные*) и *фактографические* (или *экстенсиональные*, *предметные*). Концептуальную часть знаний, то есть базу знаний, обычно называют *моделью предметной области,* а фактографическую – *базой данных*.

Процедурные знания – это совокупность процедур (алгоритмов, программ), выполняющих преобразования информации и решающих заранее определенные конкретные задачи.

Метазнаниями называют знания о свойствах знаний и методах их использования.

В некоторых системах, основанных на знаниях, различают факты и правила. При этом *факты* рассматриваются как элементарные единицы знаний – простые утверждения о свойствах объектов, а *правила* используются для выражения связей между отдельными фактами или их комбинациями.

Основным вопросом при разработке систем, основанных на знаниях, является выбор модели знаний – способа формализации и внутримашинного представления знаний. Наиболее распространенными и изученными моделями знаний являются логические, продукционные, сетевые и фреймовые.

Разработка систем, основанных на знаниях, в сущности, сводится к решению двух фундаментальных проблем [11]. Первая из них заключается в определении состава знаний, то есть в поиске ответа на вопрос «что представлять». Важность решения данной задачи объясняется тем, что из ее неудовлетворительного решения следует неадекватность создаваемых моделей. Вторая основная проблема заключается в способе представления знаний, в ответе на вопрос «как представлять», поскольку выбранный способ представления знаний может оказаться непригодным либо неэффективным.

Представляется достаточно очевидным, что эти проблемы взаимосвязаны, и вторая проблема должна решаться после решения первой.

В свою очередь, проблема выбора способа представления знаний может быть разбита на две подзадачи: определение некоторого формализма (способа структуризации знаний) и представление знаний в выбранном формализме.

Таким образом, разработка систем, основанных на знаниях, требует решения следующих задач:

- определения состава знаний;
- выбора способа организации знаний;
- определения модели представления;
- использования выбранного представления.

6.4. Проблема представления

Обработка любой информации на ЭВМ в конечном итоге может рассматриваться как преобразование входной последовательности символов в другую последовательность – выходные данные [22]. Число типов данных и операций, непосредственно выполняемых машиной, крайне ограничено с целью упрощения аппаратных средств. Поэтому для решения конкретной задачи на ЭВМ требуется соответствующее представление данных и процедур их преобразования.

Проблемой представления называют проблему перехода от внешнего представления любых объектов и процедур во внутреннее по отношению к ЭВМ. Эта проблема заключается в установлении соответствия между внутренними символами ЭВМ, с одной стороны, и объектами, межобъектными отношениями и операциями с объектами, с другой стороны. Выбор представления определяет, какие виды обработки данных могут быть автоматизированы, а какие – нет.

Организация традиционной (алгоритмической) обработки данных в целом не адекватна представлениям специалистов той или иной проблемной области. ЭВМ «не знает» решения прикладной задачи в том смысле, что может выполнять лишь элементарные операции со значениями двоичных разрядов – битов, объединяемых в последовательности фиксированной длины. Таким образом, умения машины сводятся к манипулированию последовательностями битов, которые могут рассматриваться как символы.

Решение же любой прикладной задачи выражается в терминах соответствующей проблемной области. В этом заключаются принципиальные различия, о которых говорилось в главе 4, между постановкой задачи, понятной машине, и постановкой задачи, понятной для человека

Различия в способах обработки данных машинами и экспертами в какой-либо области становятся особенно заметными при автоматизации процессов, свойственных человеческому мышлению. Решение задач экспертом в конкретной проблемной области возможно вследствие использования им огромных объемов специальных знаний. Часть такой информации может быть неполной. Процессы получения решений экспертами часто малопонятны или непонятны для неспециалистов. Но если требуется построить машинную

модель решения задач экспертом, то необходимо получить модели хотя бы основных способов его рассуждений. Традиционные методы создания программного обеспечения из-за используемых в них способах представления данных и процедур непригодны для моделирования задач подобного рода.

Любой человек обладает большим запасом знаний о реальном мире, в котором он живет. Прежде всего, всю совокупность человеческих знаний можно разделить на знания общеизвестные и знания профессиональные, приобретаемые человеком в процессе специального обучения и практической деятельности.

Характерной особенностью знаний об объектах является их структуризация, упорядочивание с использованием родовидового отношения. Очевидное достоинство любой классификации или таксономии состоит в том, что с ее помощью удается значительно сократить объем сведений, подлежащих запоминанию, так как достаточно помнить характеристики класса объектов, а не каждого объекта. Указанное свойство классификации возможно вследствие присущей ей транзитивности. Системы классификации строятся по принципу «от общего – к частному», когда каждый вид объектов наделяется всеми свойствами родового объекта и характеризуется дополнительными специфическими свойствами.

Свойством транзитивности обладают и другие отношения: агрегации, каузальные (то есть отношения типа «причина – следствие»), функциональные, принадлежности множеству, некоторые виды семантических отношений и т. д.

Как отмечалось выше, вторым типом знаний являются правила. Существование правил дает возможность, в частности, устанавливать свойства объектов, не представленные в системе классификации явным образом. Именно правила позволяют приписывать виду все свойства родового понятия.

6.5. Традиционное представление знаний

При разработке программного обеспечения всегда приходится решать проблему распределения функций между человеком и компьютером. При традиционном программировании на человека возлагается в первую очередь решение задачи установления соответствия между символами ЭВМ и объектами моделируемой предметной области, то есть проблема реализации некоторой математической (формальной) модели.

Обычное решение состоит в том, что каждому классу объектов ставится в соответствие уникальное имя индексированной переменной, то есть все множество объектов фиксированного типа рассматривается как массив. Имя переменной (массива) соответствует классу объектов, а индекс указывает на один из экземпляров этого класса, то есть индивидный объект. Таким образом, возможные значения индекса идентифицируют каждый из этих объектов. Наряду с массивами данных при моделировании объектов со сложной внутренней организацией для их представления в машинной памяти используются структуры.

Модели отношений между уже представленными объектами строятся аналогичным образом: с каждым типом отношения связывается уникальное имя

переменной, а индекс указывает на конкретный кортеж в этом отношении. В случае моделирования межобъектных отношений задача усложняется тем, что в отношениях нередко участвует несколько классов объектов. Моделируемые отношения могут быть как отношениями между абстрактными объектами (классами или типами объектов), так и отношениями между индивидными объектами.

Если данные представлены во внешней памяти, то с каждым типом объектов (или классом) может быть связано имя файла, а с каждым конкретным объектом – номер записи. Если файл предназначен для хранения некоторого отношения, то это отношение определяется совокупностью полей в произвольной записи этого файла. Экземпляры отношений могут идентифицироваться также с помощью значений некоторых полей записи.

В любом случае при традиционном программировании разработчикам программного обеспечения в процессе представления объектов приходится учитывать множество фактов, не имеющих непосредственного отношения к решаемой задаче и связанных с необходимостью ее изложения на языке ЭВМ. К таким фактам относятся тип переменной, вид представления индекса или размерность массива, допустимые значения индекса и т. п. Программист должен следить за используемыми именами переменных, значениями индексов и тому подобными чисто техническими деталями внутреннего представления объектов и отношений.

Применение традиционных программ (процедур) также связано с известными неудобствами и ограничениями. Рассмотрим их на примере начисления зарплаты. Если известны стоимость единицы продукции (допустим, детали) и количество созданных работником в течение месяца единиц продукции, то легко определить его месячный заработок. С этой целью можно использовать функцию на языке C++, фрагмент которой может иметь вид:

for (i =1; i ≤ nrab; i++) zarpl[i] = a * ndetal[i];

Заметим, что в этом фрагменте неявным образом с помощью индекса i устанавливается отношение между работником, количеством произведенных им деталей и его зарплатой. Данное отношение было создано программистом при написании программы.

Традиционные программы в процессе своей работы не могут создавать новые классы объектов и отношений, они способны порождать лишь новые экземпляры объектов или элементы отношений, как это было показано во фрагменте программы. Это первый серьезный недостаток традиционных программ.

Второй их недостаток заключается в том, что они способны давать ответы только на заранее предусмотренные вопросы. Представленный выше фрагмент функции готовит ответ на вопрос о заработной плате всех работников. Но чтобы узнать, сколько продукции изготовил конкретный работник, если известен его заработок и цена единицы продукции, в тексте программы при ее разработке необходимо предусмотреть соответствующие вычисления.

Таким образом, поведение программы предопределено, и она может давать ответы лишь на предусмотренные при ее разработке вопросы. Однако далеко не

всегда можно предусмотреть все вопросы и все потенциальные ситуации. Рассмотрим в качестве примера игру в шахматы. Белые могут начать игру одним из 20 возможных ходов, на что черные также могут ответить одним из 20 ходов. Подобное резкое увеличение числа возможных состояний с каждым ходом белых или черных называют комбинаторным взрывом. Просчитать все эти варианты и запрограммировать – абсолютно безнадежное намерение. Опытные шахматисты не осуществляют перебор всех возможных вариантов, а выбирают некоторую стратегию. После оценки ситуации на доске ими выбирается подходящая, с их точки зрения, стратегия и уже в соответствии с ней осуществляется перебор ограниченного числа вариантов.

Еще одним существенным компонентом программной обработки данных является управляющая структура, под которой можно понимать структуру программы в целом. Традиционная программа представляет собой совокупность последовательно выполняемых машинных команд (или операторов языка высокого уровня), в которую в нужных местах включены команды перехода (ветвления алгоритма) и итеративные блоки. Все, что может произойти при обработке данных, должно быть предусмотрено в программе, а любая непредвиденная ситуация может привести к ее аварийному завершению.

Принцип полной детерминированности характерен даже для программ, выполняемых в режиме реального времени. Выполнение такой программы может быть прервано (приостановлено) в любой ее точке каким-либо внешним событием. При поступлении прерывания вся информация о текущем состоянии программы (о значениях всех переменных и выполняемой команде) сохраняется, и управление передается программе обработки прерываний. После ее завершения восстанавливается состояние прерванной программы, и она продолжает работу с точки, в которой произошло прерывание. Таким образом, даже обработка прерываний является программно-управляемым процессом.

Поведение человека в подобных ситуациях отличается от описанного. Если происходит «прерывание» – какое-либо внешнее событие, требующее его внимания, то человек выполняет соответствующие действия, но к предыдущему занятию может и не вернуться. Это означает, что в поведении человека текущая информация играет более активную роль. Поэтому говорят, что человеком управляют данные, и такое поведение часто считается более эффективным.

Таким образом, при традиционной обработке на ЭВМ активную роль играют процедуры, а данные являются лишь пассивным компонентом процесса обработки или даже его «страдательным» элементом.

Другие недостатки и ограничения традиционного подхода к решению задач на ЭВМ проявляются при обработке знаний, в частности, при логическом выводе и неточных рассуждениях.

Любая программа выводит новые знания на основе уже имеющихся. Более того, при традиционном подходе считается неоспоримым предположение, что порядок логического вывода должен быть заранее известен, что, собственно говоря, и позволяет представить процесс вывода в виде традиционного алгоритма. Приведенный выше пример с вычислением заработной платы является примером логического вывода, но только одного значения, поскольку

процесс заранее определен. В реальной действительности возникают разные ситуации, и далеко не всегда ясно, что будет определено как исходные данные, и что потребуется вывести.

Рассмотрим пример, заимствованный из [22]. Пусть известны следующие отношения:

мать (A, B);
сестра (B, C);
мать (C, D);
сестра (D, E),

где «мать (A, B)» равносильно «A является матерью B». Из заданных отношений можно вывести другие родственные связи между женщинами, например, «мать (A, C)», «тетя (B, E)», «бабушка (A, D)» и ряд других.

Очевидно, что эти родственные связи могут быть выражены другими наборами отношений, например, дочь (B, A) и т. п. Важно также то, что при традиционном подходе единственным способом получения ответов на любой запрос является предварительное порождение всех возможных отношений, что повлечет за собой потребность в дополнительных ресурсах ЭВМ. Причиной возникновения подобных проблем служит недостаточная гибкость традиционных управляющих структур.

В главе 2 было дано самое общее представление о многозначной логике. В реальной жизни также часто встречаются ситуации, когда те или иные утверждения или факты нельзя однозначно охарактеризовать как ложные или как истинные. В подобных ситуациях об истинности высказываний можно говорить лишь с некоторой долей вероятности. Тем не менее, на основе неточных или неполных знаний можно делать вполне определенные заключения или выводы. Но традиционные управляющие структуры не подходят для решения таких задач. При использовании правил подобные задачи решаются сравнительно легко: одни правила устанавливают вероятности известных фактов, а другие правила осуществляют вывод в соответствии со значениями вероятностей.

Выше знания, используемые при машинном решении задач были разделены на следующие:

- факты и отношения, характерные для предметной области;
- процедуры и правила манипулирования фактами;
- управляющие структуры, содержащие сведения об условиях и способах применения указанных процедур и правил.

Таким образом, чтобы переложить на ЭВМ часть функций, выполняемых человеком при традиционном подходе к решению задач, необходимо найти лучшие способы:

- *представления объектов и отношений*, по возможности, без привлечения специальных знаний из области информатики;
- *представления правил*;
- *разработки* и использования соответствующих *управляющих структур*.

6.6. Общие принципы альтернативного представления

Альтернативный подход к представлению фактов и отношений основывается на применении формальной теории логики, в частности, на исчислении предикатов. По существу (или в абстрактном плане), проблема представления фактов сводится к определению имен объектов, а представление некоторого отношения – к определению взаимосвязи между различными объектами через имя, присвоенное данному отношению, называемому предикатом. Тот факт, что объект *Y* является компонентом объекта *X*, может быть представлен как

содержит (X, Y)

или как

содержится (Y, X).

Интерпретация данных выражений зависит от того, что собой представляют объекты X и Y. Если это абстрактные объекты (понятия), то эти выражения следует понимать в том смысле, что объекты типа Y являются компонентами объектов типа X. Если X и Y – индивидные объекты, то данные выражения означают, что конкретный объект Y входит в состав конкретного объекта X.

С помощью предикатов можно выражать любые отношения между абстрактными и эмпирическими объектами. Так, выражение

водоем (озеро, пруд, водохранилище)

устанавливает отношение между абстрактными объектами, и его интерпретация заключается в том, что озера, пруды и водохранилища являются разновидностями водоемов. Иными словами, данный предикат устанавливает родовидовое отношение между абстрактными понятиями. Выражение

река (Амур, Волга, Дон, Енисей)

устанавливает отношение между абстрактным понятием реки и конкретными эмпирическими объектами, то есть с помощью данного предиката осуществляется идентификация объектов. Данное выражение можно трактовать как отношение принадлежности множеству, если под символом «река» усматривать множество денотатов, то есть множество эмпирических объектов, попадающих под определение реки. По этой причине отношения подобного типа иногда называют классифицирующими.

Использование предикатов приобретает еще большую выразительность, если наряду с переменными в качестве аргументов используются еще и символы логических связок: $\neg$ (отрицание), $\wedge$ (конъюнкция), $\vee$ (дизъюнкция) и $\rightarrow$ (импликация), обозначаемые также соответственно как НЕТ, И, ИЛИ и ЕСЛИ…ТО. Например, можно установить соответствие между предикатами «содержит» и «содержится» следующим образом:

содержится (Z, X) ЕСЛИ содержит (X, Z)

ИЛИ (содержит (X, Y) И содержится (Z, Y)).

Это выражение должно интерпретироваться следующим образом: «содержится (Z, X) есть ИСТИНА, если истинно содержит (X, Z), или одновременно истинны содержит (X, Y) и содержится (Z, Y)». Таким образом,

каждое правило должно включать перечень переменных, к которым оно применяется. При этом предполагается, что правило применимо ко всему множеству значений входящих в него переменных.

Особенностью способа записи правил в виде предикатов является то, что они не содержат никаких специальных знаний из области программирования и определяются в форме, принятой для классов и отношений. Таким образом, правила – это отношения, в которых набор утверждений после символа ТО имплицируется утверждениями, представленными после ЕСЛИ.

При обработке знаний удобными часто оказываются продукционные системы, включающие три вида компонентов: классы и отношения; правила; управляющую структуру. Совокупность классов и отношений трактуется как база данных, содержащая декларативные знания. Процедуры представляют собой наборы правил вида «ЕСЛИ (условие) ТО (действие)». Управляющая структура определяет порядок выполнения правил. (Условие) определяет вид проверки состояния базы данных, а (действие) устанавливает некоторые действия, если состояние базы данных отвечает (условию). При выполнении этих действий состояние базы данных может измениться.

Применение правил демонстрируется на примере нахождения наибольшего и наименьшего значений в некотором наборе чисел (рис. 6.1).

В данном примере база данных представляет собой массив A из 100 чисел, n – число элементов массива, Amin и Amax – наименьшее и наибольшее значения, i – текущее значение индекса. Каждое правило применяется или, как говорят, «срабатывает» только тогда, когда не срабатывает ни одно предыдущее правило и когда выполняется условие после слова if. «Срабатывание» означает выполнение действий после слова then. Прави-ла 2 и 3 изменяют содержимое базы данных, обновляя значения переменных Amin и Amax. Правило 4 «срабатывает» только тогда, когда не срабатывают правила 1, 2 и 3. Правило 1 «срабатывает» только один раз, завершая обработку. Writeln () – процедура, выводящая на печать значения своих аргументов. Функции управляющей структуры сводятся к поочередному применению правил до тех пор, пока не будет выполнен оператор stop.

База данных			
A_1	A_2	…	A_{100}
a_1	a_2	…	a_{100}
n = 100 *Amin* = A_1 *Amax* = A_1 *i* = 1			

Правила (процедуры)
1. ***if*** *i* > *n* ***then*** *writeln (Amin, Amax)*; ***stop***;
2. ***if*** *A(i)* > *Amax* ***then*** *Amax* = *A*(*i*);
3. ***if*** *A(i)* < *Amin* ***then*** *Amin* = *A(i)*;
4. ***if*** нет других правил ***then*** *i* = *i* + 1;

Управляющая структура
Правила опробуются поочередно, пока одно из них не сработает. После этого процесс начинается с начала. Процесс заканчивается, если встречен оператор **stop**.

Рис. 6.1. Продукционная система

Для продукционной системы на рис. 6.1 порядок применения правил является существенным. На рис. 6.2 приводится пример продукционной системы, в которой правила могут выполняться в *произвольном порядке*.

База данных			
A_1	A_2	…	A_{100}
a_1	a_2	…	a_{100}
n = 100 *Amin* = A_1 *Amax* = A_1 *i* = 1			

Правила (процедуры)
1. ***if*** *(i* ≤*n)* ***and*** *(A(i)* > *Amax)* ***then*** *Amax* = *A(i)*;
2. ***if*** *(i* ≤*n)* ***and*** *(A(i)* < *Amin)* ***then*** *Amin* = *A(i)*;
3. ***if*** *i* > *n* ***then*** *writeln (Amin, Amax);* ***stop***;
4. ***if*** *((i* ≤*n)* ***and*** *(A(i)* ≤*Amax)* ***and*** *(A(i)* ≥*Amin))* ***then****i* = *i* + *1;*

Управляющая структура
Правила опробуются произвольно, пока одно из них не сработает. После этого процесс повторяется. Процесс заканчивается, если встречен оператор ***stop*.**

Рис. 6.2. Система правил, не зависящих от порядка

Если разработана система правил, которые могут выполняться в любой последовательности, то функциональные возможности такой программы могут расширяться простым добавлением правил, что никак не затрагивает управляющую структуру и, следовательно, не требует перепрограммирования. Указанное свойство правил является очень серьезным преимуществом систем, использующих правила, перед традиционными программами. Но создание системы правил, независимых от порядка выполнения, является довольно трудной задачей. Это возможно тогда, когда условие «*ЕСЛИ*» является единственным в

каждом правиле. При большем числе правил соблюдение указанного требования является сложной проблемой.

Стратегия, реализуемая управляющей структурой, существенным образом зависит от проблемной области. При поиске необходимых фактов и правил наиболее распространенными и известными являются стратегии, называемые поиском в глубину и поиском в ширину. Рассмотрим их применение на примере отношения агрегации.

Пусть конкретное межобъектное отношение агрегации задано графом, представленным на рис. 6.3. Его следует понимать так, что объект нижнего уровня является компонентом объекта на верхнем уровне. Допустим, нас интересует вопрос, является ли некоторый объект *X* прямо или косвенно компонентом объекта *A*. Поиск ответа на поставленный вопрос может осуществляться двумя очевидными способами.

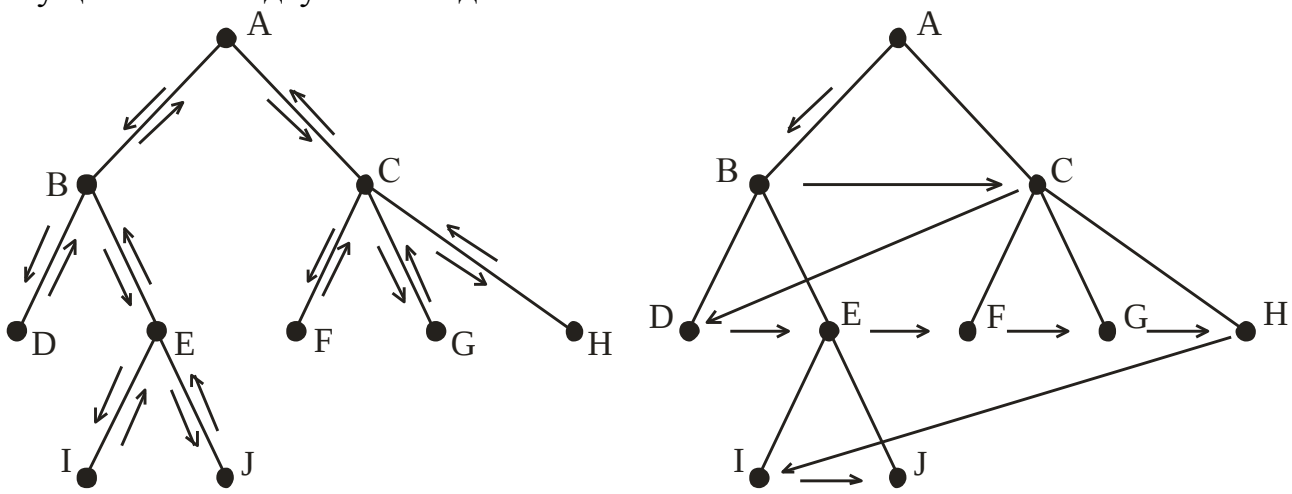


Рис. 6.3. Поиск в глубину и ширину

Первый из них заключается в том, что сначала движение осуществляется по самой левой ветви вниз до тех пор, пока не будет встречен объект X либо не будет достигнут конец ветви. Если объект X был встречен, то задача решена и ответ положительный. Если же достигнут самый нижний узел данной ветви и объект X не обнаружен, то необходимо отступить в последнюю точку ветвления и аналогичным образом проверить другие ветви. Изложенный способ называют поиском в глубину (или «сначала вглубь»), на рис. 6.3 он представлен слева. На этом же рисунке справа представлен поиск в ширину («сначала в ширину»).

Очевидно, что последовательный перебор всех правил не самый лучший способ действий и требуются стратегии направленного поиска, когда по мере получения ответов на некоторые вопросы происходит сужение области дальнейших поисков.

Во многих задачах выбор правил зависит от определенного сочетания условий или от контекста. Поэтому правила должны разумным образом группироваться. Пусть, например, решается задача распознавания картографического изображения и выделен некоторый его элемент коричневого цвета. Поскольку коричневым цветом на топографических картах изображаются только элементы рельефа, постольку далее следует анализировать только правила, относящиеся к изображению рельефа.

Замечательным свойством правил является то, что с помощью одних правил можно указывать следующее правило или совокупность правил. Таким образом, правила могут быть разбиты на два класса или уровня. Правила верхнего уровня, устанавливающие порядок применения правил нижнего уровня, называют метаправилами, то есть правилами применения правил. Использование метаправил позволяет существенным образом упростить управляющую структуру.

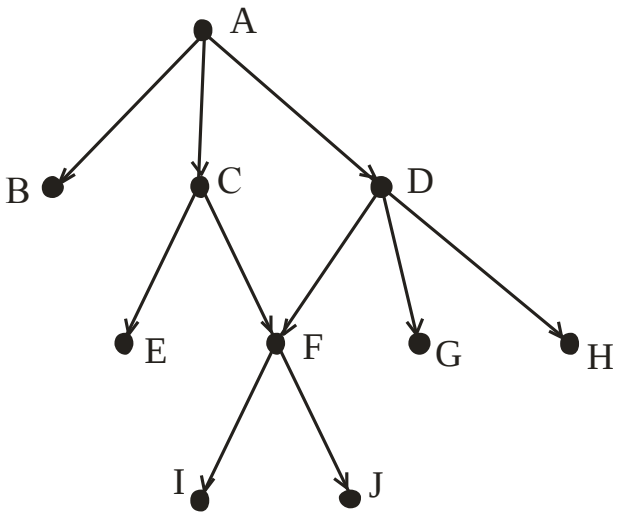


Рис. 6.4. Пример сети правил

В тех случаях, когда отношения между правилами фиксированы и известны, целесообразно объединять их в сети. Пример такой сети изображен на рис. 6.4, на котором каждый узел соответствует некоторому правилу, а стрелка между узлами X и Y означает правило ЕСЛИ X, ТО Y.

6.7. Представление знаний с помощью предикатов

Можно считать, что логика непосредственно связана с программированием, поскольку любую программу (последовательность преобразований символов) можно рассматривать как последовательность квазилогических утверждений, определенным образом преобразующих входные данные в результаты.

В логике существуют точные способы для определения истинности произвольного утверждения: утверждение считается истинным тогда и только тогда, когда истинны все относящиеся к нему предположения. Суждение об истинности конкретного утверждения может быть вынесено только после его проверки. В рамках логики проверка утверждения сводится к его сопоставлению с некоторыми определенными абстрактными моделями утверждений и выбору релевантной модели утверждения. Такие модели утверждений трактуются как «формы», состоящие из абстрагированных фактов и правил, истинность которых ранее была строго (формально) доказана.

Рассмотрим следующий пример. Допустим, существует правило «Если Иванов изучает логику, то Иванов любит точность», и стал известен простой факт «Иванов изучает логику». На основании этого факта и упомянутого правила можно сделать очевидное заключение «Иванов любит точность».

Формально этот вывод можно представить с помощью логической модели:

Если А, то В. А. Следовательно, В.

В данной схеме А и В означают соответственно утверждения «Иванов изучает логику» и «Иванов любит точность». Структура заключения об Иванове совпадает с определенной логической структурой, поэтому можно сделать

вывод о справедливости заключения «Иванов любит точность». Рассуждая подобным образом, мы фактически заменили приведенные в «форме» символы (A и B) «содержанием» утверждений.

Содержание конкретных утверждений логикой не изучается. Вполне возможно, что конкретные высказывания не имеют смысла в рассматриваемой предметной области. Так, суждение «Если Иванов изучает логику, то грядет глобальное потепление климата» может оказаться бессмысленным. Но, как выразился один из американских авторов, в действиях наблюдаемого логики может быть больше, чем кажется наблюдающему. Не исключено, что Иванов решил заняться изучением причин такого потепления. Важно то, что с точки зрения логики заключение «Если Иванов изучает логику, то грядет глобальное потепление климата. Иванов изучает логику. Следовательно, грядет глобальное потепление климата» является справедливым, даже если оно и абсурдно.

Логическая модель «Если A, то B. A. Следовательно, B.», или «форма», представляет собой рассмотренное ранее правило «modus ponens», являющееся одним из основных в логике. Данное правило тесным образом связано с продукционными системами для обработки знаний, в которых оно используется следующим образом. Пусть в базе знаний среди множества правил имеется некоторое правило «ЕСЛИ P, ТО Q». Пусть также в базе данных продукционной системой зафиксирован некий факт P. Тогда система, обнаружив совпадение факта P с левой частью приведенного правила, сгенерирует утверждение Q в качестве нового факта и поместит его в базу данных, в которой он первоначально не присутствовал. Далее продукционная система может применить другое правило, левая часть которого совпадает с Q. В итоге работа простой продукционной системы сводится к многократному применению правила «modus ponens». Таким образом, подобные системы обладают способностью анализировать имеющиеся и порождать новые факты предметной области.

При использовании исчисления предикатов для представления знаний отношениям ставятся в соответствие предикаты, а объектам и другим понятиям предметной области – их аргументы. Структура предикатов имеет вид *имя_предиката (аргумент_1,…, аргумент_n).* В качестве имен предикатов используются имена свойств и отношений, а в качестве аргументов – константы и переменные предметной области. Константы могут обозначать конкретные индивидные объекты либо классы объектов. При использовании в качестве аргументов переменных конкретные индивидные объекты или классы таких объектов остаются незаданными. Переменные принимают значения, когда им присваивается значение константы – имя конкретного индивидного объекта или класса объектов. Например, выражения

площадной объект (X),

водный путь (X, Y)

интерпретируются соответственно как «(неизвестный) объект *X* является площадным» и «имеется водный путь между *X* и *Y*». Когда переменной ставится в соответствие некоторая константа, происходит порождение экземпляра этой переменной.

Порядок следования аргументов предикатов и их интерпретация устанавливаются разработчиками и должны быть постоянными в пределах продукционной системы.

Элементарные высказывания, состоящие из предикатов и связанных с ними аргументов, с помощью логических операций (пропозициональных связок) могут объединяться в сложные высказывания. Примеры сложных высказываний:

– *является (Джомолунгма, высочайшая вершина)* $\wedge$ *находится (Джомолунгма, Непал)*, то есть «Джомолунгма является высочайшей вершиной и находится в Непале»;

– *водный путь (A, B)* $\wedge$ *период судоходства (время)* $\rightarrow$ *использовать (доставка грузов, A, B, водный транспорт)*, то есть «если существует водный путь между пунктами *A* и *B*, и время приходится на период судоходства, то для доставки грузов из *A* в *B* следует использовать водный транспорт».

В классическом исчислении предикатов для манипулирования переменными используются кванторы. Кванторы устанавливают определенные требования к экземплярам переменных с тем, чтобы высказывание в целом было истинным. Для квантора общности все значения переменной, обозначающей некоторый класс или тип объектов, должны быть «истинны»:

$\forall$*X (дорога федерального значения (X)* $\rightarrow$ *имеет (X, твердое покрытие))*, что следует понимать как «все дороги федерального значения имеют твердое покрытие».

Для квантора существования истинными могут быть только некоторые значения переменной:

$\exists$*X (река (X)* $\rightarrow$ *пересыхает (X, засушливый период)*, то есть «некоторые реки в засушливый период пересыхают».

Использование исчисления предикатов позволяет представить многие сложные предложения естественного языка без присущей последнему неоднозначности. Кроме того, такое представление может осуществляться в форме, удобной для обработки на ЭВМ. Так, высказывания:

образован (Новосибирск, 1893) $\wedge$ *число жителей (Новосибирск, 1,5 млн.)*, то есть «Новосибирск образован в 1893 г. и число его жителей равно 1,5 млн.»;

качество воды (озеро, пресная) $\rightarrow$ *использовать (озеро, источник водоснабжения)*, то есть «если вода в озере пресная, то оно может использоваться как источник водоснабжения»,

могут быть преобразованы в эквивалентную форму:

$\neg$ ($\neg$ *образован (Новосибирск, 1893)* $\vee$ $\neg$ *число жителей (Новосибирск, 1,5 млн.),* то есть «неверно, что Новосибирск образован не в 1983 г. или что число его жителей не равно 1,5 млн.»;

$\neg$ *качество воды (озеро, пресная)* $\vee$ *использовать (озеро, источник водоснабжения)*, то есть «вода в озере не пресная или оно используется как источник водоснабжения».

Полученные после преобразований высказывания полностью *равносильны* исходным как по смыслу, так и по истинностным значениям, но более трудны

для понимания. Приведенные высказывания кажутся противоречивыми, и человек должен обладать определенным навыком для их правильного понимания. Поскольку ЭВМ не обращается к смыслу высказываний и выполняет все операции над высказываниями по формальным правилам исчисления предикатов, не изменяющим их истинностных значений, постольку появляется возможность выбора формы представления конкретных высказываний в виде, наиболее удобном для манипулирования ими.

6.8. Логический вывод

Применение исчисления предикатов для обработки знаний имеет принципиальное значение в связи с возможностью логического вывода новых фактов и новых правил из некоторого числа исходных.

Рассмотрим в качестве примера набор следующих фактов:

1) род (инженерная сеть, трубопроводы);
2) род (трубопроводы, газопровод);
3) род (газопровод, ацетиленопровод)

и два правила:

1) $\forall$(X, Y) (род (X,Y) $\rightarrow$ вид (Y, X));
2) $\forall$(X, Y, Z) (род (X, Y) $\wedge$ вид (Z, Y) $\rightarrow$ вид (Z, X)),

которые можно проинтерпретировать следующим образом:

1) для всех *X* и *Y*, если объект *X* есть родовой объект по отношению к *Y*, то объект *Y* является разновидностью объекта *X*;
2) для всех значений *X*, *Y* и *Z*, если *X* есть родовой объект относительно *Y* и *Z* является видом *Y*, то *Z* является разновидностью *X*.

Теперь предположим, что требуется выяснить, является ли ацетиленопровод разновидностью инженерных сетей. Ни одно из имеющихся правил не позволяет непосредственно ответить на поставленный вопрос. Но на основании факта 3 и правила 1 можно получить промежуточное высказывание

вид (ацетиленопровод, газопровод).

Применив правило 2 к полученному факту и факту 2, выводим новый факт

вид (ацетиленопровод, трубопровод).

Теперь на основании этого факта и факта 1, а также правила 2 находим, что

вид (ацетиленопровод, инженерная сеть).

В действительности для получения этого результата мы применяли не только вышеназванные правила 1 и 2, называемые правилами предметной области, но и два «оставшихся в тени» правила более высокого уровня, называемые правилами вывода. Первым из таких правил является известное правило modus ponens, которое записывается также как

$$A \rightarrow B, A \mapsto B,$$

где символ $\mapsto$ трактуется как «следовательно». Второе примененное нами в неявном виде правило – это правило «специализации»

$$\forall(X)\,\Phi(X), A \mapsto \Phi(A),$$

которое выражает интуитивные представления о том, что если некоторый тип объектов характеризуется каким-либо свойством P, то свойство P присуще всем объектам данного типа. Это правило можно рассматривать как подстановку значения константы A вместо переменной X. Такая подстановка будет вполне правомерной, если A принадлежит области значений переменной X.

Правила вывода представляют собой утверждения самого общего вида о связях между посылками и заключениями, которые в исчислении предикатов всегда справедливы. Очевидно, что правила предметной области применимы только в ее пределах, что и подчеркивается в их названии. Правила вывода можно представлять как элементы более высокого уровня абстракции, на котором рассматривается задача манипулирования правилами предметной области. Общим свойством правил вывода в исчислении предикатов является их универсальная истинность. Поэтому они могут применяться как для установления истинности конкретных утверждений в целом, так и для порождения заключений. Правила вывода могут применяться отдельно либо совместно с другими. К основным правилам вывода при обработке знаний наряду с modus ponens и правилом специализации относят также правила:

- modus tollens: $A \to B, \neg B \mapsto \neg A$;
- двойного отрицания: $A \mapsto \neg(\neg A)$;
- введения конъюнкции: $A, B \mapsto (A \wedge B)$;
- приведения к абсурду (или противоречию): $A \to B, A \to \neg B \mapsto \neg A$.

Выше было показано, что, используя факты, правила предметной области и правила вывода, можно получать доказательства или опровержения тех или иных утверждений. Но при автоматизации процессов логического вывода возникает ряд проблем. Одна из них вызвана отсутствием четких критериев при выборе фактов и правил: какие правила применять, когда применять и к каким фактам применять.

Другая проблема связана с различной природой логического вывода в исчислении предикатов и ходом рассуждений, осуществляемых человеком и основанных на здравом смысле. Особенность классического исчисления предикатов состоит в том, что в нем никогда не аннулируется заключение, если становятся известными дополнительные факты. В частности, если была доказана истинность импликативного утверждения $A \to B$, то какое-либо новое истинное утверждение C не изменяет его истинности, что можно записать как $(A \wedge C) \to B$. Данное свойство исчисления предикатов называют монотонностью. Говорят, что из монотонности классического исчисления предикатов следует, что любые доказательства в нем аддитивны и их пересмотр не требуется.

Иначе обстоит дело в реальной действительности, когда человек вынужден изменять полученные ранее заключения. Так, человек может сделать вывод, что для передвижения на автомобиле можно использовать полевую дорогу, если глубина брода не превышает 0,5 м, то есть сделать заключение

глубина (брод, менее 0,5 м) → использовать (полевая дорога); глубина (брод, менее 0,5 м) ⊢ использовать (полевая дорога).

Если стал известен дополнительный факт

поднялась (река, 2 м),

то есть вода в реке поднялась на 2 м, то высказывание

(глубина (брод, менее 0,5 м) ∧ поднялась (река, 2 м)) → использовать (полевая дорога)

человек уже не будет считать справедливым. Данный пример показывает принципиальное отличие рассуждений, основанных «на здравом смысле», от логического вывода в исчислении предикатов: их немонотонность.

Вообще-то, проблему немонотонности рассуждений, основанных на здравом смысле, можно связать с проблемой полноты и корректности знаний. Очевидно, что если бы в базе знаний присутствовало правило относительно пользования полевыми дорогами при поднявшихся реках, то не пришлось бы его пересматривать. Просто наше правило

глубина (брод, менее 0,5 м) → использовать (полевая дорога),

как показала жизнь, оказалось неполным и, как следствие, некорректным. Поэтому мы сделали вывод о немонотонном характере рассуждений, осуществляемых человеком. Казалось бы, дополнив существующее правило условием, относящимся к уровню воды в реках, мы могли бы получить правило, более адекватное реалиям мира, в котором мы живем. И таким образом снять проблему немонотонности рассуждений.

Но… мир постоянно изменяется, и те правила, что были верны вчера, завтра могут утратить свою истинность. Допустим, что в некотором гипотетическом мире есть компьютеры, но нет воздушного транспорта. Предположим также, что в некой базе знаний хранится правило:

требуется (скорость) → использовать (железнодорожный транспорт).

Данное правило будет справедливым в этом мире до изобретения самолета, после чего правило потребуется откорректировать и придать ему вид:

требуется (скорость) → использовать (воздушный транспорт).

Можно было бы пытаться предвидеть некоторые изменения и ввести в базу знаний следующие правила:

требуется (скорость) → использовать (самый быстрый транспорт);

самый быстрый транспорт (железнодорожный транспорт).

Но наши ухищрения не дали бы результата, поскольку последнее правило все равно пришлось бы модифицировать. Таким образом, проблема немонотонности рассуждений является достаточно сложной. Как сказал один из авторов, реальный мир сложен, потому что конкретен. А конкретность является едва ли не синонимом уникальности.

Автоматизация логического вывода требует определенной стратегии применения правил вывода. Последовательный перебор всех правил вывода в надежде, что какое-либо из них сработает, может потребовать неприемлемых затрат машинного времени. Это связано с тем, что в процессе логического вывода может потребоваться вывод дополнительных фактов в предположении,

что они будут использованы на более поздних этапах доказательства. В процессе логического вывода существует также угроза комбинаторного взрыва, при котором порождается бесконечное множество высказываний вида $A \wedge B \wedge C \wedge C \wedge ... \wedge C$ и т. п.

Поэтому машинная реализация доказательств методами исчисления предикатов требует разработки набора процедур для выбора правил, которые бы исключали возможность комбинаторного взрыва.

При решении этой проблемы выделяются два подхода. Первый подход заключается в отказе от принципа универсальности и поиске таких методов обработки данных, которые были бы эффективны при решении конкретных задач в определенной проблемной области. Поставленная цель достигается введением в систему больших объемов знаний о фиксированной предметной области и следованием логике экспертов, решающих задачи на основе этих знаний. Для упрощения процессов программирования при разработке подобных систем были созданы определенные формализмы, обладающие меньшими возможностями, чем исчисление предикатов, но более удобные с точки зрения их машинной реализации. При этом в центре исследований оказываются методы представления фактов и процедурных знаний в виде, который позволяет выявить аспекты, имеющие принципиальное значение в конкретной предметной области. Определение методов представления знаний – это только один класс проблем при разработке систем, основанных на знаниях.

Другая совокупность проблем сводится к управлению знаниями. Первый подход может рассматриваться как ликвидация различий между правилами предметной области и правилами вывода. Собственно говоря, в исчислении предикатов указанное разделение правил осуществляется только по соображениям обеспечения универсальности. В системах, основанных на знаниях, правила предметной области и правила вывода заменяются общим набором правил, универсальных только по отношению к конкретной предметной области. Разрабатываемые при этом управляющие структуры по необходимости отражают структуру знаний о предметной области. Отсюда следует, что сфера применения подобных систем ограничивается исключительно конкретной предметной областью.

Второй подход в определенной мере является альтернативным первому и основывается на традиционной логике и сохранении присущего исчислению предикатов свойства универсальности, когда разрабатываются мощные универсальные процедуры доказательств, исключающие как проблему выбора правил вывода, так и проблему комбинаторного взрыва. В основе этого подхода лежит один из методов доказательства теорем, названный методом резолюции.

При использовании метода резолюции все выражения исчисления предикатов сначала приводятся к упрощенной форме записи, называемой клаузальной формой. Любые логические связки могут быть выражены с помощью других логических связок. Такой процесс упрощения сводится к редуцированию выражений к перечню предикатов, соединенных знаком дизъюнкции $\vee$. Кроме того, используется символ импликации, поскольку требуются определенные средства для представления правил в явном виде.

6.9. Экспертные системы

Экспертной системой (ЭС) называют вычислительную систему, включающую знания специалистов из некоторой проблемной области и способную получать решения задач из этой области, качество которых сравнимо с качеством решений, получаемых экспертами. Фундаментальным отличием экспертных систем от традиционных программных комплексов является переход от обработки данных к обработке знаний, в связи с чем их часто называют также системами, основанными на знаниях. Таким образом, в экспертных системах вместо традиционного сочетания «данные + алгоритм = программа» реализована новая архитектура программного обеспечения «знания + вывод = система», которую иногда называют революционной [21].

В качестве отличительных признаков экспертных систем чаще всего указывают следующие характеристики, вытекающие из их архитектуры:

1) ориентация на конкретную область применения;
2) получение не количественных, а качественных результатов;
3) четкое разделение знаний и механизма вывода;
4) преимущественное использование правил;
5) способность делать заключения при неточных или недостаточных данных;
6) способность объяснения полученных выводов;
7) наличие средств адаптации и развития системы.

Наиболее типичные системы, основанные на знаниях, содержат компоненты, представленные на рис. 6.5:

1) *базу знаний* – механизм представления знаний о конкретной проблемной области;

2) *решатель задач* – механизм логического вывода на основании знаний, имеющихся в базе знаний;

3) *пользовательский интерфейс*, осуществляющий функцию обмена информацией между пользователем и системой;

4) *механизм передачи знаний* от квалифицированного специалиста системе;

5) *механизм объяснения решений*, полученных системой;

6) *базу данных* – описание текущего состояния проблемной области.

В действующих системах, основанных на знаниях, некоторые из указанных компонентов могут быть расширены дополнительными функциями, некоторые могут находиться в вырожденном виде, но основные структурные элементы – база знаний и механизм логических выводов – присутствуют всегда.

База знаний содержит факты – некоторые утверждения о состоянии предметной области и правила. Правила содержат информацию о том, как порождать гипотезы или новые факты из имеющихся. Время жизни фактов меньше, чем время жизни правил, и они играют более пассивную роль, чем правила. Знания не встроены в текст программы, как это имеет место в традиционных программах, а представляют собой данные для интерпретатора,

каким является машина вывода. Знания представляются обычно в виде системы продукций (правил), семантических и фреймовых сетей.

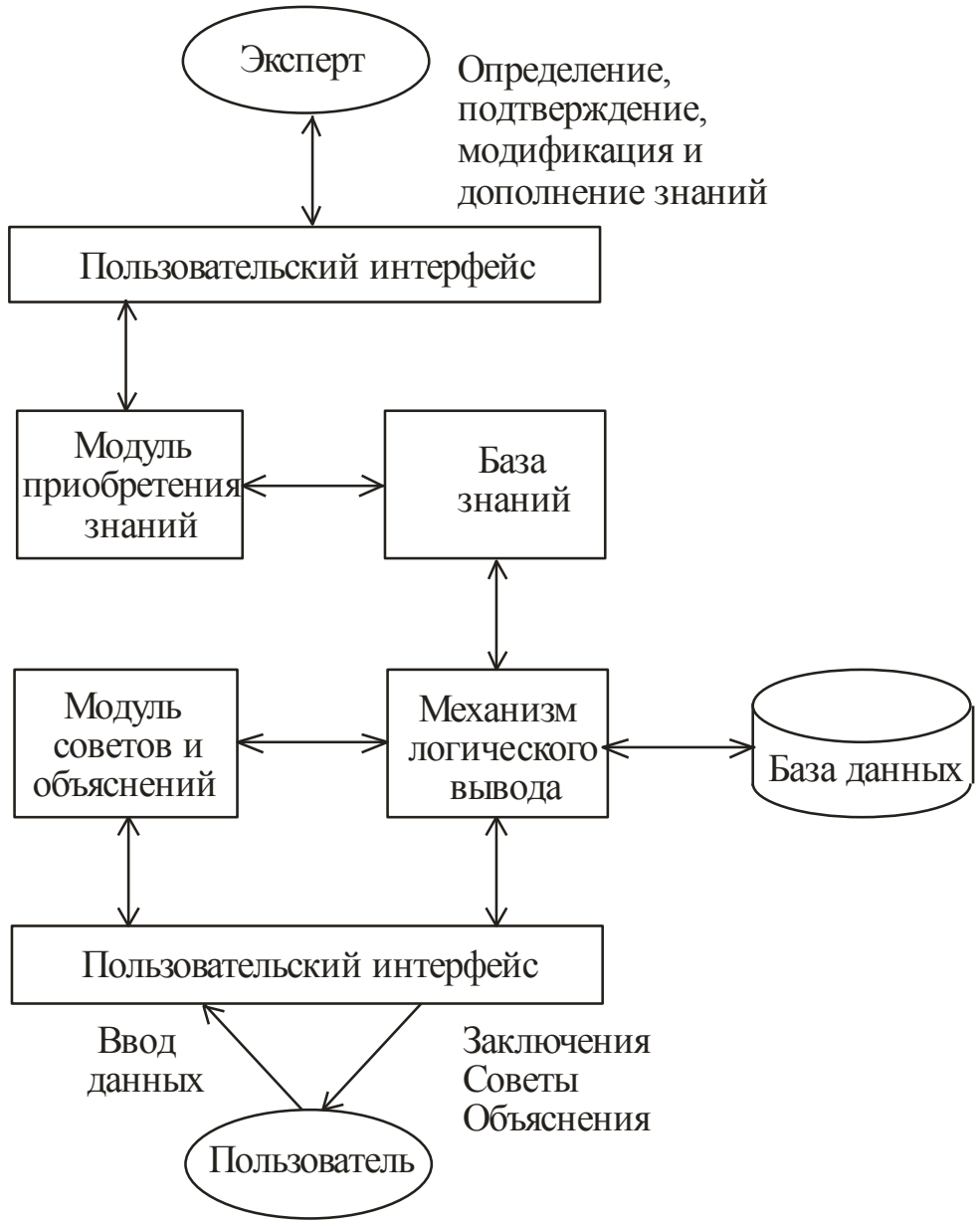


Рис. 6.5. Структура экспертной системы

Механизм логического вывода является, наряду с базой знаний, обязательным компонентом ЭС и существенным образом отличается от традиционных программ с их жесткими алгоритмами. Его функционирование в целом определяется состоянием базы данных. Удаление хотя бы одного оператора или команды из традиционной программы означает ее катастрофу. При удалении же части правил из базы знаний качество решений, получаемых экспертной системой, несколько снизится, но система полностью сохраняет свою работоспособность.

Различия между экспертными системами проявляется в стратегии логического вывода. В одних системах реализован прямой вывод (от фактов к гипотезам), в других – обратный (от гипотез к подтверждающим их фактам). Наиболее эффективными являются системы с двунаправленным выводом.

Некоторые авторы считают наличие *механизма объяснения решений* принципиальной особенностью систем, основанных на знаниях. Но, скорее всего, значимость данного компонента ЭС определяется их проблемной областью и назначением. Если ЭС предназначена для обучения, то реализация механизма объяснения решений является обязательной уже в силу назначения системы.

Пользовательский интерфейс присутствует всегда, иначе общение с системой было бы невозможно. Когда говорят о пользовательском интерфейсе, то имеют в виду его развитость и удобство с точки зрения пользователя. Удобство пользовательского интерфейса и наличие компоненты объяснения решений определяют дружественность системы по отношению к пользователю.

Замечательной особенностью систем, основанных на знаниях, является их способность функционировать в двух режимах (приобретения знаний и решения задач), а также то, что размер области решаемых задач или применимость системы зависит не столько от длины программы, сколько от объема базы знаний. Кроме того, существуют системы, предназначенные для обучения, например, в области медицины и т. п.

Очевидно, что экспертная система должна содержать знания из предметной области и что только этих знаний недостаточно. Принято считать, что состав знаний зависит от проблемной области, структуры экспертной системы, требований пользователей и используемого языка представления знаний.

В соответствии со схемой функционирования ЭС выделяют следующие знания:

- знания о процессе решения задач, используемые интерпретатором;
- знания о языке общения и способах организации диалога с пользователем, необходимые для пользовательского интерфейса;
- знания о способах представления знаний, используемые механизмом приобретения знаний;
- структурные и управляющие знания, используемые объяснительной компонентой.

Классификация знаний с точки зрения архитектуры ЭС представлена на рис. 6.6.

Рис. 6.6. Структура знаний ЭС

Неинтерпретируемыми называют знания, содержание и структура которых не «осознается» интерпретатором или другой компонентой системы, все остальные знания относят к *интерпретируемым*.

Неинтерпретируемые знания подразделяют на вспомогательные и поддерживающие знания. *Вспомогательные знания* используются естественно-языковой компонентой для ведения диалога с пользователем, и их содержанием являются знания о лексике и грамматике языка общения и описание структуры диалога. *Поддерживающие знания* используются в процессе разработки системы и при объяснении ее действий и разделяются на технологические и семантические. *Технологические знания* содержат сведения об авторах и времени создания знаний и т. п. *Семантические знания* включают смысловое описание технологических знаний, их назначение и причины создания, способ использования и другие сведения, которые могут оказаться полезными для разработчиков системы.

Среди интерпретируемых знаний выделяют знания о представлении, управляющие знания и предметные знания. Знания о представлении и

управляющие знания выполняют роль метазнаний по отношению к предметным знаниям. *Знания о представлении* содержат описание структур, с помощью которых осуществляется представление знаний в системе.

Управляющие знания подразделяют на фокусирующие и решающие. *Фокусирующие знания* предназначены для сужения области поиска решений и представляют собой предписания об использовании знаний в тех или иных ситуациях, сведения о наиболее вероятных гипотезах и о знаниях, используемых при проверке соответствующих гипотез в первую очередь. В первом случае внимание системы концентрируется на содержимом рабочей памяти, а во втором – на содержимом базы знаний. *Решающие знания* – это знания о выборе способа интерпретации знаний, релевантных текущей ситуации. Таким образом, управляющие знания создаются и используются с целью повышения эффективности вывода.

Предметные знания включают данные о сущностях, свойствах сущностей и отношениях между сущностями предметной области, а также о способах преобразования этих данных. В предметных знаниях принято выделять описатели и собственно предметные знания. *Описатели* характеризуют собственно предметные знания и указывают такие особенности последних, как коэффициент определенности правил и данных, степень их сложности или важности. *Собственно предметные знания* – это совокупность фактов и операционных знаний. *Факты* трактуются как элементарные или составные утверждения о сущностях предметной области, а *операционные знания* представляют собой сведения о том, каким образом может изменяться описание предметной области в процессе функционирования системы.

Характеристики экспертной системы улучшаются при использовании метазнаний. Метазнания не представляют собой единого блока и рассредоточены в системе. Метазнания могут использоваться в экспертной системе с целью:

выбора релевантных правил;

обоснования применения правил из области экспертизы;

анализа синтаксической и семантической корректности предметных правил;

адаптации системы к предметной области и др.

Иногда осуществляют более крупное подразделение знаний на знания стратегические, структурные и поддерживающие. При этом к *стратегическим* относят знания о плане, в соответствии с которым в процессе решения задачи выполняется упорядочивание целей и гипотез. Стратегические знания эффективно представляются с помощью метазнаний, не зависящих от предметной области. Таким образом, стратегические правила обеспечивают механизм для применения правил.

Структурные знания трактуются как знания, уточняющие стратегические знания. Структура, представляющая правила, может быть очень сложной. Между посылками и окончательным заключением может быть несколько промежуточных заключений. Структурные знания используются для навигации в сети правил.

Поддерживающие правила используются для обоснования переходов от посылок к заключениям. Среди них различают идентифицирующие правила, причинные правила, факты мира и факты области экспертизы.

Идентифицирующие, или *определительные, правила* основываются на свойствах того или иного класса и используют идентифицирующие свойства объектов для их классификации. *Причинными* называют правила, в которых предпосылки и заключения связаны причинными отношениями. Представление причинных правил может быть направлено как от причины к следствию, так и в обратном направлении. *Факты мира* – это специфические знания о предметной области, основанные на здравом смысле. Например, высота верха любого объекта не может быть меньше высоты его низа. *Правила о фактах области* выражают закономерности предметной области и могут использоваться для вывода одних свойств из совокупности других свойств.

Одной из первых экспертных систем была система DENDRAL, созданная в 1965 г. в Станфордском университете и предназначенная для обработки масс-спектрометрических данных. Ее задачей являлся вывод возможных структур молекул по химическим формулам и масс-спектрограммам. Среднее качество решений, предложенных системой, оказывалось выше среднего качества решений специалистов. Эта система продемонстрировала потенциальные возможности искусственного интеллекта, что имело большое теоретическое и практическое значение. Однако известные до сих пор принципы не могут быть обобщены, поскольку нередко сводятся к совокупности здравых рассуждений.

Среди других ЭС можно указать:

– MYCIN, предназначенную для диагностики и лечения бактериальных инфекций крови и ставшую родоначальницей целого ряда медико-диагностических систем (TEIRESIAS, EMICIN, PUFF…);

– PROSPEKTOR, предназначенную для решения геологических проблем и открывшую ранее неизвестное месторождение молибдена;

– R1, областью применения которой было проектирование конфигурации ЭВМ VAX-11/780;

– HEARSEY-III, представлявшую собой комплекс программных средств для построения экспертных систем вне зависимости от области применения последних.

Экспертная система EURISKO отличается от других ЭС тем, что она является самообучающейся. В ней используется индуктивный вывод, что позволяет ей улучшать и расширять запас собственных эвристических знаний. Эта система три года подряд одерживала победу в учебных военных играх, хотя правила игры каждый год менялись с целью поставить ее в затруднительное положение. Эта же система изобрела трехмерный узел типа И/ИЛИ, что повлекло за собой переворот в области сверхбольших интегральных схем. Поскольку передача знаний от экспертов системе считается очень сложной задачей, постольку разработка самообучающихся ЭС представляет собой чрезвычайно перспективное направление в области систем, основанных на знаниях [21].

6.10. Модели представления знаний

Представление знаний является центральной проблемой при разработке систем, основанных на знаниях. Считается, что если данная проблема в конкретной системе решена удачно, то остальные вопросы решаются «сами собой».

Из различий между декларативными и процедурными знаниями («знать что» и «знать как») следуют различия в их представлении. Представление процедурных знаний основано на том, что интеллектуальные способности машины определяются знаниями, вложенными в программы, то есть знаниями о том, как использовать свойства сущностей.

Декларативное представление базируется на относительной независимости сущностей и процедур, используемых для их обработки. При использовании декларативного представления предполагается, что интеллектуальные способности машины являются следствием существования некоторого универсального множества процедур, способных обрабатывать данные любого типа, и наличия множества специфических фактов, в совокупности отражающих состояние конкретной проблемной области. Преимущество декларативного представления по сравнению с процедурным представлением состоит в том, что те или иные декларативные знания могут использоваться разными способами. Может оказаться неэффективным заранее фиксировать способы использования фактографических знаний, то есть утверждений об объектах предметной области. Более разумным может оказаться решение отложить этот вопрос до того времени, когда тот или иной факт окажется в центре внимания системы информационного моделирования. Данное свойство декларативных знаний позволяет придать их представлению определенную гибкость и экономичность за счет разных способов использования одних и тех же фактов.

Декларативные знания слабо зависят не только от процедурных знаний. Обычно одни декларативные знания слабо зависят от других декларативных знаний. Эта их особенность позволяет разбивать все множество декларативных знаний на сравнительно независимые или слабо связанные утверждения о свойствах моделируемых объектов. Отсюда следует относительная легкость создания, модификации и контроля достоверности декларативных знаний.

Создание и модификация процедурных знаний являются более сложной проблемой, так как приходится учитывать способ использования утверждений каждого типа. Однако существуют такие сущности, которые можно легко описать в виде процедуры и довольно трудно в декларативном представлении. Примером может служить хотя бы календарь. В принципе, для указания дня недели на любую дату в прошлом и будущем можно воспользоваться декларативным представлением и составить таблицу достаточно большого размера. Но понятно, что такое представление потребует больших трудозатрат и повлечет за собой неэффективное использование памяти ЭВМ, и что в данном случае лучше использовать процедурное представление знаний.

Поскольку нельзя говорить о бесспорных преимуществах использования декларативных или процедурных знаний, постольку естественно стремление к использованию сочетания положительных качеств обоих способов. Результатом подобных стремлений явились формальные методы, использующие смешанное представление: или так называемое декларативное представление с присоединенными процедурами, или процедурное представление в виде модулей с декларативными образцами.

Все множество формальных методов представления знаний, называемых *моделями знаний*, обычно разбивают на два типа: *логические* и *эвристические*. В свою очередь, эвристические модели подразделяются на *семантические*, *фреймовые* и *продукционные*.

В логическом направлении были получены существенные результаты в представлении знаний с использованием логики предикатов первого порядка и найдены универсальные и достаточно эффективные способы логического вывода.

Эвристические модели в искусственном интеллекте развивались представителями когнитивного направления, опиравшимися в своих исследованиях на принципы организации человеческой памяти и моделирование вывода на основе таких принципов. Результатам, полученным при когнитивном подходе к представлению знаний, присуща в большей мере выразительность, чем математическая строгость. Но если представители логического направления объектом своего изучения выбирали сравнительно простые задачи, то в когнитивном направлении были получены решения достаточно сложных реальных задач, потребовавшие использования новых методологических подходов.

С течением времени исследования в логическом направлении завершились решением ряда задач, поставленных сторонниками нелогического направления, поэтому в 1980-х гг. было пересмотрено отношение к использованию логических методов, и они стали применяться в эвристических моделях представления знаний.

В логических моделях представления знаний используются те или иные формальные системы. Среди таких систем наибольшее применение находит исчисление предикатов первого порядка. Его использование особенно усилилось после разработки таких универсальных и эффективных методов дедуктивного вывода, как метод резолюций и обратный метод. Особенностью логических моделей знаний является получение вывода из заданной системы посылок с применением фиксированного множества правил вывода.

Особенность систем, использующих индуктивный вывод, состоит в том, что конкретные знания об отношениях внутри каждой предметной области не задаются пользователем, а создаются системой. В логических системах индуктивного типа правила генерируются самой системой на основе некоторого конечного числа обучающих примеров. Хотя индуктивные модели представления знаний широкого распространения не получили, они представляют безусловный интерес при разработке интеллектуальных картографических систем.

Эвристические модели представления знаний отличаются как большим разнообразием, чем логические модели, так и более высокой эффективностью, поскольку являются специализированными.

Системы, в которых знания представляются совокупностью правил вида «*если – то*», получили название *продукционных систем.* Среди таких систем различают системы двух диаметрально противоположных типов: с прямым и обратным выводом.

Логические выводы подразделяют на прямые, обратные и двунаправленные. В системах с обратным выводом с помощью правил строится дерево И/ИЛИ, связывающее в единое целое исходные факты и заключения. Оценка полученного дерева на основании представленных в базе данных фактов и есть логический вывод. В системах с прямым выводом отправной точкой служат исходные данные, а заключением является гипотеза, соответствующая самому верхнему узлу дерева, то есть корню. Вывод в них заканчивается в узлах с отрицанием. Однако для прямого вывода характерно использование большого количества данных и оценок, не имеющих никакого отношения к заключению. Обратный вывод считается более эффективным, поскольку в нем оцениваются только факты, имеющие отношение к заключению, но если определенное отрицание или утверждение невозможны, то порождение дерева не имеет смысла. При двунаправленных выводах процесс начинается с оценки небольшого объема имеющихся данных, после чего выбирается гипотеза и осуществляется поиск данных, подтверждающих ее правильность. Системы, использующие двунаправленный вывод, считаются более мощными и гибкими.

Первые продукционные системы основывались на системах продукций с прямым выводом. Поэтому они являются «основополагающими» в том смысле, что содержат основные концепции продукционных систем. В этих системах четко выделяются три обязательных компонента: база данных, база правил, состоящая из множества продукций, и механизм логического вывода (интерпретатор) на основе совокупности правил вывода. Вывод осуществляется в виде цикла «понимание – выполнение», и если некоторое правило оказывается применимым к ситуации (состоянию базы данных), то состояние базы данных изменяется в соответствии с правой частью примененного правила. В результате последовательности подобных преобразований состояние базы данных трансформируется из исходного в целевое. Таким образом, продукционная система характеризуется простым циклом выборки и выполнения правил. Однако на каждом шаге вывода требуется сопоставление с образцом в базе правил, так называемое отождествление, поэтому с увеличением числа правил время вывода существенно возрастает и такие системы оказываются непригодны для решения больших задач.

Таким образом, преимущества продукционных систем заключаются в простоте:

- создания и понимания правил;
- расширения и модификации правил;
- механизма логического вывода.

Недостатками же продукционных систем считаются:

- сложность понимания взаимных отношений правил;
- сложность восприятия всей совокупности знаний;
- низкая эффективность обработки;
- отсутствие гибкости в осуществлении вывода;
- отличие структуры от структуры человеческих знаний.

Фреймовая модель представления знаний основана на фреймовой теории М. Минского, являющейся систематизированной теорией психологической модели памяти человека и его сознания. Но, из-за отсутствия конкретных идей по реализации языка представления знаний только на основе этой теории, она считается достаточно абстрактной. Основополагающим в этой теории является понятие фрейма как некоторой структуры данных для представления концептуальных (абстрактных) объектов. Каждый фрейм рассматривается как совокупность слотов, содержащих определенную информацию. Все фреймы могут быть связаны между собой, в результате чего образуется единая фреймовая система, естественным образом объединяющая декларативные и процедурные знания.

Теория фреймов инициировала несколько попыток разработки языков представления знаний, которые позднее получили довольно широкое распространение в области искусственного интеллекта.

Моделью семантической сети называют модель представления знаний, представляющей собой семантическую сеть. Общепризнанного определения семантической сети нет, но чаще всего под ней понимают систему знаний, образ которой ассоциируется с сетью, состоящей из узлов (понятий и объектов) и дуг (отношений между узлами). Так, если имеются узлы «объект *X*», «метеостанция» и дуга «есть», соединяющая эти узлы, то этой конструкции можно поставить в соответствие факт «объект *X* есть метеостанция». Факты подобного типа могут быть представлены иначе, если ввести с целью унификации узел «есть», и с каждым узлом связать один их типов «объект», «понятие» и «отношение».

6.11. Логические модели

Такие формальные системы и теории, как исчисление высказываний и исчисление предикатов, лежащие в основе логических моделей, строятся с использованием элементарных объектов и весьма ограниченного числа правил оперирования этими объектами. Так, в некоторых логических системах единственным правилом вывода является правило modus ponens. Поэтому в логических моделях представления знаний отношения между отдельными элементами знаний выражаются только с помощью того минимального арсенала средств, которые предоставляются используемой формальной теорией.

Основой логических моделей является формальная теория T, представляющая из себя четверку

$$T = (B, F, A, R),$$

где B – конечное множество символов, называемых алфавитом теории T и используемых для образования конечных последовательностей, называемых выражениями теории T;

F – подмножество выражений теории T, имеющих смысл и называемых формулами T;

A – фиксированное подмножество формул, называемых аксиомами теории T, то есть априорно истинными формулами данной теории;

R – конечное множество отношений $R = \{r_1, ..., r_n\}$ между формулами, называемое правилами вывода. Для каждого r_i существует целое положительное число k такое, что для каждого множества k формул и для каждой произвольной формулы f может быть решен вопрос о том, находятся или нет заданные k формул в отношении r_i с формулой f. Если отношение r_i выполняется, то f называют непосредственным следствием данных формул по правилу r_i. Следствием или выводом формулы f_n в теории T называют последовательность формул $f_1, ..., f_n$, в которой каждая формула f_i $(i = 1, ..., n)$ является либо аксиомой теории T, либо получена как непосредственное следствие из предшествующих формул по одному из правил вывода.

Правила вывода позволяют по мере необходимости расширять множество формул, являющихся истинными в теории T.

Формальная теория называется разрешимой, если существует единая процедура, позволяющая для любой формулы f установить, существует или нет ее вывод в T. Формальную теорию T называют непротиворечивой, если формулы f и $\bar{f}$ одновременно не выводимы в T.

Наиболее часто для создания логических моделей представления знаний используется исчисление предикатов первого порядка.

Алфавит исчисления предикатов образуют следующие символы:

1) пропозициональные связки $\{\neg, \wedge, \vee, \rightarrow\}$;
2) символы кванторов $\{\forall, \exists\}$;
3) символы предметных констант – строчные первые буквы латинского алфавита с индексами или без них;
4) символы предметных переменных – строчные последние буквы латинского алфавита, при необходимости – с индексами;
5) n-местные функциональные буквы – строчные буквы греческого алфавита, при необходимости – с верхними и нижними индексами { $\varphi_k^n, k \geq 1, n \geq 0$ и т. п.}, символы вида φ_k^0 используются также для обозначения констант;
6) n-местные предикатные символы – прописные средние буквы латинского алфавита без индексов или с индексами { $P_k^n, k \geq 1, n \geq 1$ и т. д.};
7) символы пунктуации {(,) .}.

Выражения исчисления предикатов подразделяются на термы, элементарные формулы и правильно построенные формулы. Простейшими выражениями являются термы – символы предметных констант или переменных. Если t1, …, tn (n ≥ 1) – термы, то f_k^n (t1, …, tn) также является термом.

Если t1, …, tn (n ≥ 1) – термы, и P_k^n – предикатная буква, то P_k^n (t1, …, tn) – элементарная формула, или атом. Все элементарные формулы являются правильно построенными формулами. Если A и B правильно построенные формулы, то $\neg A$, $A \vee B$, $A \wedge B$ и $A \to B$ суть правильно построенные формулы. Если A – правильно построенная формула и x – переменная в A, то $(\forall x)A$ и $(\exists x)A$ – правильно построенные формулы. В выражениях $(\forall x)(A)$ и $(\exists x)(A)$ A называют областью действия соответственно квантора общности и квантора существования. Переменная x называется связанной, если она находится в области действия квантора, примененного к переменной. Если к переменной не применяется квантор, то она называется свободной. Так, в формуле $\forall x(P(x, y) \to Q(x))$ переменная x является связанной, а переменная y – свободной. Формулу, не содержащую свободных переменных, называют замкнутой.

Формулы наполняются содержанием при их интерпретации как некоторых утверждений о предметной области. Интерпретацией называют систему, состоящую из непустого множества D, называемого областью интерпретации, и некоторого соответствия, относящего каждой предикатной букве P_k^n некоторое n-местное отношение в D, каждой функциональной букве φ_k^n – некоторую n-местную функцию, отображающую $D^n \to D$, а каждой константной букве φ_k^0 – некоторый элемент из D. При этом предполагается, что при заданной интерпретации переменные «пробегают» область D этой интерпретации. Всякой элементарной формуле при заданной интерпретации приписывается значение 1 (истина) или 0 (ложь). Элементарной формуле P_k^n (t1, …, tn) приписывается значение 1, если и только если термы предикатной буквы соответствуют элементам из D, удовлетворяющим отношению, определяемому данной интерпретацией.

Истинностное значение неэлементарных формул может вычисляться рекуррентно как значение истинностной функции от значений составляющих ее формул. Если A и B – формулы, то значения формул $\neg A$, $A \vee B$, $A \wedge B$ и $A \to B$ устанавливаются по таблицам истинности.

Формулам вида $(\forall x)(A)$ соответствуют утверждения: «для любого значения x из области интерпретации D значение формулы A истинно», а формулам $(\exists x)(A)$ – утверждения «существует некоторое значение x из области D такое, что значение формулы A истинно». Приведенные утверждения

могут быть как истинными, так и ложными. Если область значений истинности таких формул конечна, то их истинность может быть установлена с помощью таблиц истинности. Конкретные формулы могут быть истинными или ложными в зависимости от выбранной области интерпретации.

Формула A называется выполнимой, если и только если существует интерпретация I, в которой формула A принимает значение 1 в I. Если формула A принимает значение 1 в интерпретации I, то говорят, что I удовлетворяет формуле A. Если формула A принимает значение 1 при всех интерпретациях, то ее называют общезначимой. Так, формула $P(a) \to (P(a) \vee P(b))$ является общезначимой, поскольку истинна при любой интерпретации. Формулу A называют невыполнимой, если при всех интерпретациях ее истинностное значение равно 0.

Формула A логически следует из формул $A_1, ..., A_n$ тогда и только тогда, когда любая интерпретация, удовлетворяющая $A_1, ..., A_n$, удовлетворяет также и A. Формулы $A_1, ..., A_n$ при этом называют посылками, а формулу A – заключением логического следования, что обозначается как $A_1, ..., A_n \mapsto A$. Доказана следующая теорема, называемая теоремой дедукции: формула A является логическим следствием формул $A_1, ..., A_n$ тогда и только тогда, когда формула $A_1 \wedge ... \wedge A_n \to A$ общезначима, то есть $\mapsto (A_1 \wedge ... \wedge A_n) \to A$.

Доказательство той или иной теоремы состоит в выяснении вопроса логического следования некоторой формулы A из заданного множества формул $A_1, ..., A_n$, что равносильно доказательству общезначимости формулы $A_1 \wedge ... \wedge A_n \to A$ или невыполнимости формулы $A_1 \wedge ... \wedge A_n \wedge \neg A$.

В исчислении предикатов первого порядка не существует общего метода установления общезначимости любых формул. Имея в виду указанное свойство, говорят, что исчисление предикатов первого порядка является неразрешимым. Если же некоторая формула исчисления предикатов общезначима, то существует процедура проверки ее общезначимости, поэтому исчисление предикатов первого порядка называют также полуразрешимым.

Основным достоинством исчисления предикатов как модели представления знаний является единообразие формальной процедуры доказательства теорем. Но высокая степень единообразия влечет за собой и основной недостаток такой модели знаний – сложность использования при доказательстве эвристик, специфичных для каждой проблемной области. Этот недостаток проявляется наиболее значимо при создании экспертных систем, эффективность которых определяется преимущественно знаниями, характерными для конкретной проблемной области. К недостаткам формальных систем относят также их монотонность, отсутствие средств структурирования используемых элементов и недопустимость противоречий.

С целью устранения указанных недостатков формальных систем как средства представления знаний были разработаны семиотические системы. Семиотическая система представляет собой восьмерку

$$S = (B, F, A, R, Q(B), Q(F), Q(A), Q(R)),$$

где первые четыре элемента те же, что и в определении формальной системы, а остальные компоненты являются правилами их изменения под влиянием накапливаемых в базе знаний сведений о строении и функционировании сущностей проблемной области. Таким образом, семиотическая система имеет два уровня: на первом (нижнем) уровне для описания модели представления знаний используется (традиционная) формальная система, а на втором (верхнем) уровне определяется модель адаптации используемой формальной системы. Иначе, средства второго уровня представляет собой метасистему или метамодель по отношению к модели первого уровня. Однако теория семиотических систем находится в начальной фазе своего развития.

6.12. Продукционные модели

Выше объяснение различий между традиционным и альтернативным представлением знаний основывалось на модели знаний, называемой системой продукций. Формально продукция может быть представлена выражением

$$(i); Q; P; A \to B; N,$$

где приняты следующие обозначения.

Символ i является уникальным именем продукции, выделяющим конкретную продукцию из множества продукций. В качестве имени может выступать лексема, выражающая суть именуемой продукции (как-то: «мост», «ремонт моста», «прилив», «хвойный лес»…), или номер продукции.

Q характеризует ситуацию или область применения продукции. С каждой ситуацией или задачей в памяти человека хранятся связанные с ней понятия и знания, которые человек извлекает по мере необходимости. Чтобы минимизировать время поиска нужной информации при решении задач на ЭВМ, подобные связи должны моделироваться тем или иным способом.

$A \to B$ называют ядром продукции. Интерпретация ядра может быть различной в зависимости от того, что стоит слева и справа от символа $\to$, но чаще всего истолковывается как «ЕСЛИ А, ТО В» или «ЕСЛИ А, ТО В1, иначе В2». Другое толкование ядра продукции состоит в том, что А обозначает некоторое условие, необходимое для выполнения действия В.

Р обозначает условие применимости ядра продукции. Обычно Р представляет собой некоторое логическое выражение, чаще всего – предикат. Ядро продукции активизируется, если значение Р есть «истина». Если Р ложно, то ядро продукции не может быть использовано.

N описывает некоторые постусловия продукции, под которыми понимают действия, которые необходимо выполнить после реализации ядра продукции.

Системы продукций могут считаться самой распространенной моделью представления знаний. Их отличительными чертами являются:

− близость к логическим моделям знаний, что дает возможность организации эффективного вывода;

− возможность представления большинства человеческих знаний в виде продукций;

− высокая модульность представления знаний, в результате чего достигается простота создания правил и управления ими;

− высокая синтаксическая однородность представления знаний, что позволяет создать сравнительно простой механизм их использования системой;

− возможность реализации любых алгоритмов, то есть возможность представления процедурных знаний;

− присущий системам продукций естественный параллелизм и асинхронность выполнения операций, что является предпосылками их использования в ЭВМ новой архитектуры для выполнения так называемых параллельных вычислений.

Продукционная система состоит из компонентов трех типов:

− набора правил;

− рабочей памяти;

− механизма логического вывода (решателя задач).

Набор правил используется как база знаний, и иногда его называют *базой правил*. *Рабочая память* используется для кратковременного хранения фактов (предпосылок и результатов вывода) предметной области. *Механизм вывода* использует правила для получения фактов, представленных в базе знаний неявным образом.

Рассмотрим работу наиболее типичной продукционной системы на следующем примере. Пусть имеется некоторая экспертная система по вопросам трудоустройства, выбора профессии, профессионального образования и смежным вопросам. Пусть в ее базе знаний среди множества правил имеются следующие.

1. *ЕСЛИ профессия – картограф И необходимо изучить перспективное направление ТО следует изучить формальную картографию.*

2. *ЕСЛИ есть работа по специальности И работа хорошо оплачивается И специальность нравится ТО НЕТ причин менять профессию.*

3. *ЕСЛИ НЕТ причин менять профессию И возраст менее 40 лет ТО цель – профессиональная карьера.*

4. *ЕСЛИ цель – профессиональная карьера ТО необходимо изучить перспективное направление.*

Предположим далее, что за консультацией обратился некий человек. Система ничего не знает о консультируемом, поэтому она будет задавать вопросы. Возможно, что первый вопрос, который задаст экспертная система, будет вопрос о специальности. Получив ответ, что он картограф, продукционная система занесет в пустую до этого момента рабочую область первый факт:

1. Профессия – картограф,

и попытается применить какое-либо правило. Для этого механизм вывода сопоставляет левую часть правила с фактами, хранимыми в рабочей области.

Если рабочая память содержит все условия некоторого правила, то его условная часть считается истинной, в противном случае – ложной.

В данном примере ни одно правило не может быть применено, поскольку в посылке правила 1 истинно только одно простое утверждение, а о необходимости изучать перспективное направление ничего не известно. Поэтому система продолжит задавать вопросы.

Допустим далее, что на все вопросы системы: «Имеете ли вы работу по специальности?», «Хорошо ли оплачивается работа?», и «Нравится ли вам профессия?» пользователь ответил утвердительно. (Здесь, конечно, имеется элемент фантастики, но не будем обращать на него внимания.). После этого в рабочую область будут занесены факты:

1. *Профессия – картограф.*
2. *Есть работа по специальности.*
3. *Работа хорошо оплачивается.*
4. *Специальность нравится.*

После занесения последнего факта в рабочую область сработает правило 2 и система сделает довольно естественный вывод «*НЕТ причин менять профессию*» и придаст ему статус нового факта. Тогда содержимое рабочей области примет вид

1. *Профессия – картограф.*
2. *Есть работа по специальности.*
3. *Работа хорошо оплачивается.*
4. *Специальность нравится.*
5. *Нет причин менять профессию.*

На этом этапе ни одно правило не применимо, но система может обнаружить, что для применения правила 3 необходимо задать вопрос о возрасте. Получив ответ «25» и сравнив его с 40, система дополнит рабочую область новым фактом

6. *Возраст менее 40 лет,*

после чего на основании правила 3 сделает вывод, который трудно оспорить и который будет помещен в рабочую область как факт

7. *Цель – профессиональная карьера.*

На основании данного факта и правила 4 система устанавливает следующий бесспорный факт:

8. *Необходимо изучать перспективное направление.*

Теперь к фактам 1 и 8 применяется правило 1 и следует вывод:

9. *Следует изучить формальную картографию.*

Других применимых правил в базе знаний нет, и система заканчивает свою работу выдачей результата вывода: «Следует изучить формальную картографию».

Ход рассуждений системы пользователю не виден, поэтому не исключено, что данная рекомендация его удивит. Действительно, не каждый человек согласится с тем, что

ЕСЛИ профессия – картограф И есть работа по специальности И работа хорошо оплачивается И специальность нравится И возраст 25 лет ТО следует изучить формальную картографию.

Поэтому пользователь вправе потребовать от системы объяснений. Но, даже получив их, пользователь может не согласиться с таким заключением. Кроме того, в данном примере инициатива при ведении диалога принадлежит системе, но это условие не более чем особенность примера. В действующих экспертных системах инициатива может переходить от системы к пользователю и обратно.

В рассмотренном примере для получения вывода данные извлекались из рабочей памяти, затем применялось правило и новые факты помещались в рабочую память. Такой вывод называют прямым выводом. Его противоположностью является обратный вывод, при котором делается попытка подтвердить факты, которые могут выступить в качестве заключения. При обратном выводе в нашем примере система выдвинула бы некоторую гипотезу. Таковой могла оказаться цель «следует изучить формальную картографию».

Система пытается подтвердить это заключение. Но в рабочей памяти находится только факт 1, а истинность утверждения «необходимо изучить перспективное направление» неизвестна. Данное утверждение является заключительной частью правила 4, поэтому в качестве новой гипотезы выбирается условная часть правила 4 «цель – профессиональная карьера» и т. д. Останов системы при обратном выводе происходит, когда достигается первоначальная цель либо когда заканчиваются правила, применимые для ее достижения в процессе вывода.

Данный пример позволяет сделать следующие выводы.

1. Разработка правил является достаточно сложной проблемой. Даже такой простой пример проблемной области требует размышлений. При разработке реальной системы первой проблемой явилось бы определение границ области экспертизы. В частности, следовало бы определить, должна ли подобная система содержать сведения об образовательных учреждениях и потребностях производственных организаций в конкретных специалистах. Если да, то при эксплуатации потребовались бы значительные усилия по поддержанию базы данных в актуальном состоянии.

2. Система правил существенным образом зависит от представлений эксперта. Другой эксперт мог, например, сформулировать правило 1 в менее спорном виде:

ЕСЛИ специальность – картограф И необходимо изучить перспективное направление ТО следует изучить теоретические основы применения ЭВМ,

либо предложить другую систему правил.

3. Система правил, насчитывающая несколько сотен или более правил, трудна для восприятия.

4. Рассуждения системы могут быть непонятны пользователю, поэтому требуется разработка объяснительной компоненты.

5. Выводы системы носят рекомендательный характер, и окончательное решение должно оставаться за пользователем.

Следует отметить, что в продукционных системах существует очень серьезная проблема. В примере с пользователем-картографом на каждом этапе было применимо только одно правило. Но обычно в действующих системах к содержимому рабочей области применимо несколько правил. Множество правил, применимых на некотором этапе логического вывода, называют *конфликтным набором*. Выбор одного из правил, составляющих конфликтный набор, называют *разрешением конфликта*. Стратегия, применяемая для разрешения конфликтов, оказывает очень сильное влияние на эффективность вывода в продукционной системе.

Кроме того, до сих пор была рассмотрена базовая структура продукционных систем, на практике же часто используются разнообразные дополнительные средства. Например, иногда требуется уточнить контекст, в котором выполняется правило. В подобных случаях в рабочую память помещается дополнительная информация, характеризующая определенную ситуацию или объект. При этом дополнительные данные представляются с помощью триплета или тройки: «объект – атрибут – значение».

Одно из преимуществ применения структуры «объект – атрибут – значение» проявляется при существовании нескольких объектов одного типа. Любое правило, применимое для объектов некоторого класса, применимо к каждому объекту этого класса, но не более одного раза. Поэтому система должна контролировать корректность применения правил. Данную проблему называют проблемой контекста применения правила, объект же применения правила называют контекстом применения правила. Другое преимущество структуры «объект – атрибут – значение» состоит в том, что ее расширение до структуры вида «объект – атрибут – значение – фактор достоверности» позволяет легко включать в систему нечеткие знания.

В системах продукций сопоставление данных из условной части правил с данными рабочей памяти является главным моментом. В большинстве случаев численные значения сравниваются по величине, но при этом важно не само значение величины, а его принадлежность к некоторому диапазону. Для описания подобных условий используется понятие предиката.

В канонических продукционных системах в условной части правил проверяются данные из рабочей области: наличие или отсутствие данных либо выполнение условий, заданных предикатами. Условная часть правил может состоять из одного условия или нескольких условий, соединенных логической связкой И. В заключительной части правил указываются данные, которые должны быть помещены в рабочую память при выполнении условной части.

На практике при разработке продукционных систем в эту принципиальную схему вводят дополнительные возможности: в условной части правил используется связка ИЛИ; вводится условная часть с вычислениями по данным из рабочей памяти; применяются правила, заключительная часть которых не изменяет содержимого рабочей памяти и т. п. Такие правила представляют собой не что иное, как отношение вывода, устанавливаемое между содержимым рабочей памяти, ссылка на которое присутствует в условной части правила, и

содержимым заключительной части правила. Наглядно такое отношение может быть представлено с помощью древовидного графа (рис. 6.7).

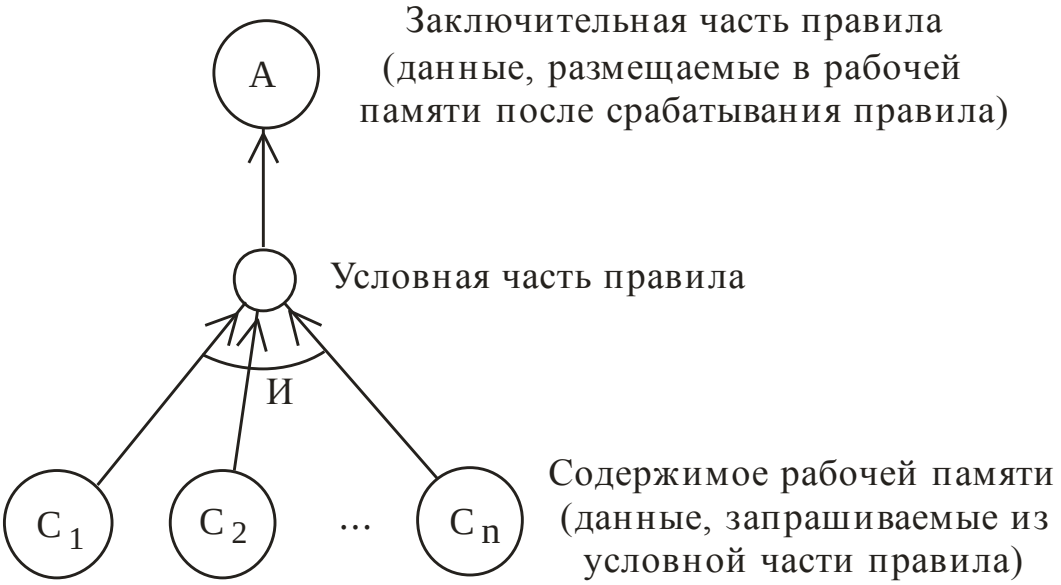


Рис. 6.7. Представление правила графом

Если существует несколько правил, из которых выводится одно и то же заключение, то данное заключение будет получено при выполнении хотя бы одного из этих правил. Поэтому данные правила можно объединить с помощью операции ИЛИ и получить правило с более сложной структурой (рис. 6.8). Следовательно, если подобным образом представить отношения между всеми используемыми правилами, то всю систему (совокупность) продукций можно представить в виде одного графа, называемого *графом И/ИЛИ*. В самых нижних вершинах такого графа будут представлены исходные данные, а в самых верхних узлах – заключения, получаемые в результате вывода. Строго говоря, такой граф нельзя считать лесом, так как каждый его подграф может быть не деревом, а сетью (рис. 6.9).

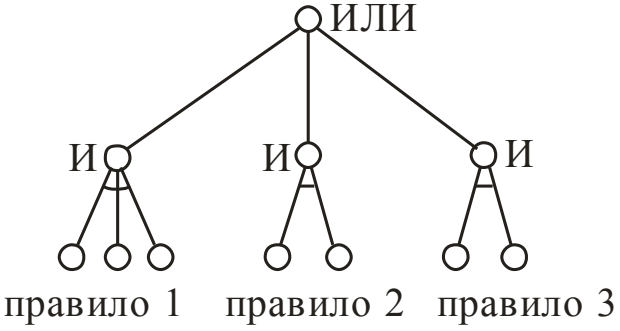


Рис. 6.8. Вывод на основе нескольких правил

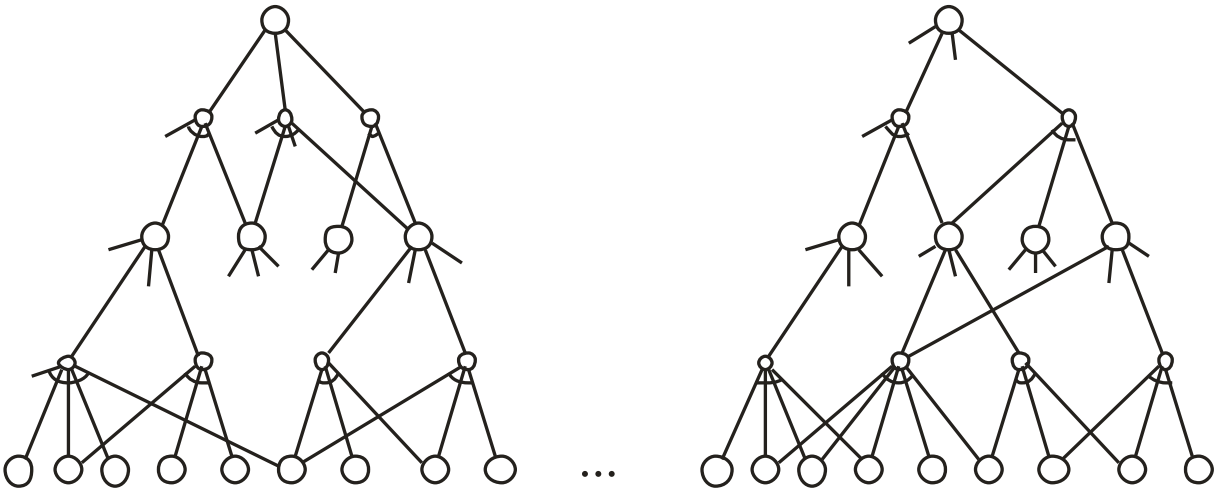


Рис. 6.9. Представление вывода в виде графа И/ИЛИ

Обратный вывод можно представить как задачу поиска пути на графе И/ИЛИ. Для этого одна из вершин верхнего уровня выбирается в качестве цели (или гипотезы). После этого анализируются все ее ветви ИЛИ. Если хотя бы одна из таких ветвей позволяет вывести поставленную цель, то доказательство этой цели (выдвинутой гипотезы) считается успешным, в противном случае – неуспешным. Выбор одной из ветвей ИЛИ графа И/ИЛИ, в сущности, является проблемой разрешения конфликтов. Определение последовательности оценки ветвей И на графе соответствует последовательности оценки отдельных условий, входящих в условную часть правила. Поэтому чрезвычайно важная с практической точки зрения проблема эффективности вывода может рассматриваться как проблема организации поиска пути на графе И/ИЛИ.

В действующих системах продукций нередко требуется иметь дело с нечеткой информацией. При этом различают два вида нечеткости: нечеткость используемых данных и нечеткость вывода. В случаях, когда заключение выводится на основе нескольких правил, некоторые из которых или все могут быть нечеткими, возникает проблема оценки нечеткости заключения или степени доверия полученному заключению.

Пусть заключение C выводится на основе множества правил $P_1, ..., P_n$. Предположим, что нечеткость заключения C характеризуется степенью доверия, которая может быть представлена некоторым числом. Каждое значение достоверности заключения, получаемого по правилу P_i, обозначим как CM_i $(i = 1, ..., n)$ (рис. 6.10.).

$$\mathrm{CM} = \mathrm{comb}(\mathrm{CM}_1, \mathrm{CM}_2, ..., \mathrm{CM}_n)$$

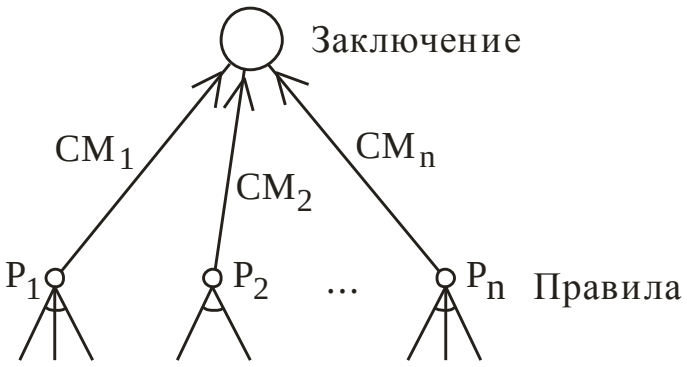


Рис. 6.10. Определение достоверности при нечетком выводе

Тогда достоверность *CM* заключения будет функцией от достоверности CM_i $(i = 1, ..., n)$ каждого правила:

$$CM = comb(CM_1, ..., CM_n).$$

Способы поиска на графе, представляющем систему продукций для случая, включающего нечеткости, зависит от вида функции *comb*. Если хотя бы одно из правил P_i является четким, то и *CM* должно иметь четкое значение, поэтому такое правило следует пытаться использовать раньше других. Более того, если достоверность заключения не вызывает сомнений, то есть характеризуется четкостью, то дальнейший поиск можно прекратить. Но если даже

- такого правила не существует,
- либо оно существует, но не может быть применено, в результате чего нельзя получить достоверный вывод,
- либо нельзя оценить достоверность всего заключения из-за неопределенной степени достоверности некоторых правил,
- итоговое заключение может быть получено, для чего следует пытаться применить все правила.

Таким образом, процесс логического вывода может быть представлен с помощью подграфов, полученных как результат поиска на всем графе И/ИЛИ. Это означает, что все правила, использованные для доказательства полученного заключения, и все данные, относящиеся к этим правилам, могут быть представлены с помощью таких подграфов, и что процесс вывода в системе может быть описан с помощью таких подграфов.

При этом имеется возможность указать правила, применяемые для получения некоторого особого заключения (последнего или промежуточного), и данные, на основании которых должны использоваться эти правила, а также указать условия, невыполнение которых влечет за собой неудачу при применении правила. Подобное представление процесса логического вывода позволяет обнаруживать ошибочные правила в тех случаях, когда

- доказательство некоторой гипотезы не было получено, хотя должно было произойти обратное,
- было получено неверное заключение.

Поэтому представление логического вывода в виде графов И/ИЛИ считается эффективным средством разработки систем, так как позволяет исследовать все возможности системы и определить направления их расширения.

Выше отмечалось, что разрешение конфликтов является важной проблемой, от успешного решения которой зависит эффективность функционирования всей продукционной системы. Следовательно, чтобы повысить эффективность работы системы, необходимо решить проблему управления последовательностью выбора правил. Кроме того, как можно было видеть на примере с пользователем-картографом, важное значение имеет состав задаваемых системой вопросов и их последовательность, поскольку от этого зависит порядок вывода.

В принципе, управление прямым выводом осуществляется проще, чем управление обратным выводом. При прямом выводе цикл обработки правил заключается в выполнении следующих действий:

- поиск данных в рабочей памяти;
- генерация конфликтного набора правил;
- разрешение конфликта (выбор правила);
- применение правила (модификация содержимого рабочей области).

При этом для управления выводом используются два способа: либо налагается ряд ограничений на генерацию конфликтного набора, либо определяется некоторый алгоритм разрешения конфликта.

Ограничения на генерацию конфликтного набора состоят в том, что, в зависимости от содержания правил, применяется один из двух методов. Первый метод заключается в том, что поиск условной части правил некоторого вида не осуществляется. Его реализация осуществляется с помощью метаправил, условная часть которых содержит требования к содержанию правил (их условной и заключительной части) и содержанию рабочей памяти, либо правил, условная часть которых указывает атрибуты (условной и заключительной частей), не подлежащие поиску или исследуемые с этой целью.

Второй метод основан на том, что множество правил предварительно разбивается на некоторые категории и в первую очередь поиск осуществляется среди правил определенной категории, зависящей от ситуации (содержания рабочей памяти). Правила сначала группируются по атрибутам и для каждой категории правил указывается условие (форма которого аналогична условной части предметных правил) относительно содержания рабочей памяти. При разрешении конфликта в первую очередь анализируется возможность применения той группы правил, в пределах которой выполняется данное условие. В других случаях разрешение или запрет на применение группы предметных правил указывается с помощью заключительной части такого правила.

При прямом выводе все заключения, которые можно получить на основе данных рабочей памяти, рано или поздно получают, так что проблема конфликтов отличается сравнительно невысокой сложностью. Самый простой способ заключается в применении правил в соответствии с порядком их определения. Поскольку каждое из релевантных правил должно быть применено, постольку при прямом выводе влияние последовательности применения правил на эффективность получения заключения может быть минимизировано.

В тех случаях, когда имеется условие останова, его достижение можно ускорить, например, применением вывода «в глубину», что фактически соответствует приоритетному применению правил со ссылкой на последние данные, поступившие в рабочую область (рис. 6.11). При таком подходе последовательность вывода минимальна и, кроме того, его можно характеризовать как способ, понятный пользователю.

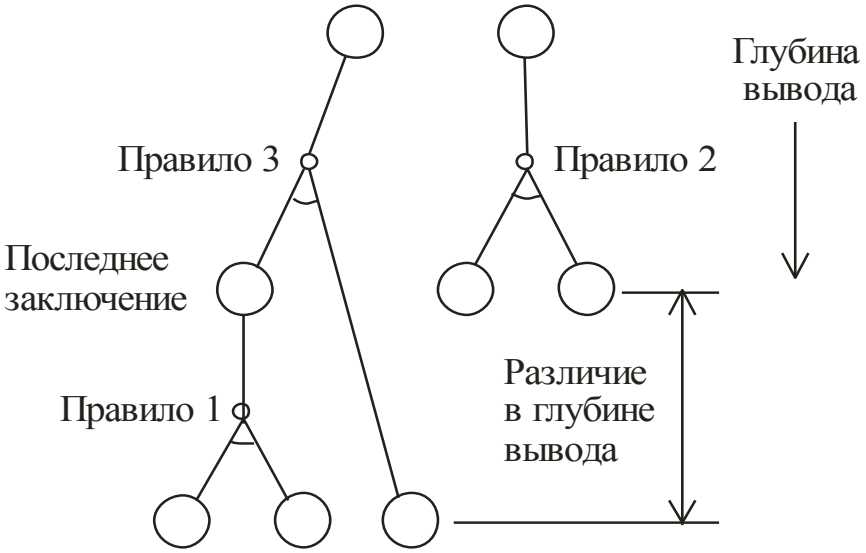


Рис. 6.11. Применение правил по приоритету

В продукционных системах чаще других используются следующие рациональные стратегии или принципы.

Принцип «стопки книг» основан на частоте использования продукций и представляет собой аналогию со стопкой, в которой наиболее нужные книги оказываются сверху. При реализации данного принципа выбираются прежде всего продукции, частота использования которых максимальна.

Принцип наиболее длинного условия заключается в выборе среди множества продукций той продукции, у которой наибольшее число условий выполнимости ядра. Данный принцип основан на том соображении, что частные правила, относящиеся к узкому классу ситуаций, содержат больше информации о конкретной ситуации, чем общие правила, относящиеся к большой совокупности ситуаций.

Принцип метапродукций предполагает создание продукций особого вида, называемых *метапродукциями*, назначением которых является управление выбором из конфликтных наборов продукций. В заключительной части метапродукций указывается очередность или предпочтительность использования продукций.

Принцип классной доски основан на аналогии с классной доской, на которой можно делать записи, а затем некоторые из них стирать. В продукционных системах «классная доска», представляющая собой область рабочей памяти, может использоваться несколькими процессами одновременно. При этом каждый процесс находит нужные данные и заносит в эту память информацию, которая может потребоваться другим процессам.

Принцип приоритетного выбора требует введения статических и динамических приоритетов продукций. Статические приоритеты создаются на основе априорной информации о важности правил, получаемой обычно от эксперта. Динамические приоритеты вырабатываются самой системой в процессе функционирования и могут отражать время пребывания продукции в конфликтном наборе, частоту использования отдельных правил, частоту следования продукций друг за другом и т. п.

Выше были изложены некоторые общие принципы, используемые стратегиями поиска вывода. В зависимости от предметной области, при прямом выводе могут разрабатываться специфические алгоритмы разрешения конфликтов.

Проблема разрешения конфликтов существует и при обратном выводе, но к ней добавляется еще проблема, связанная с последовательностью оценки условий в условной части правил. При обратном выводе возможность применения правила не означает возможность принятия решения, так как возможно, что фигурирующие в его условиях данные должны быть подтверждены с помощью вывода. Таким образом, выбор условия эквивалентен образованию новой ветви поиска, а изменение последовательности поиска негативно сказывается на его эффективности и усложняет понимание процесса вывода.

Увеличение количества правил в продукционной системе до нескольких сотен требует повышения ее эффективности. Доступный способ оптимизации поиска информации состоит в упорядочении содержимого рабочей памяти в виде структуры «объект – атрибут – значение». Существует способ, в соответствии с которым отношения между правилами и данными или между правилами и правилами предварительно представляются в виде графа и при каждом дополнении данных отыскивается ветвь графа, и связанные с ней вершины отмечаются.

Сильно гипертрофируя действительное положение дел, можно сказать, что система, использующая продукционную модель, состоит из одного оператора вида «*ЕСЛИ условие ТО действие*», называемого *правилом продукции*.

Под *действиями* в правой части правила продукции подразумеваются самые разнообразные действия, выполняемые системой или пользователем. Обычно они заключаются в изменении состояния базы данных, но могут быть обращениями к пользователю с просьбой ввода дополнительных данных или условий, обращениями к внешним устройствам и т. п.

Основными компонентами продукционных систем являются:

- *база знаний, содержащая правила продукций;*
- *база данных, отражающая текущее состояние предметной области;*
- *управляющая структура (в терминах программирования – интерпретатор), определяющая порядок применения правил и инициирующая выполнение необходимых действий.*

В сущности, множество правил продукции определяет совокупность допустимых на каждом шаге элементарных преобразований, в результате выполнения которых осуществляется продвижение от начального состояния (исходных условий) к решению задачи. Текущее состояние предметной области отражается в базе данных в виде множества фактов. В процессе решения задачи выполняется сопоставление левой части правил продукции с содержимым базы данных. В реальных продукционных системах возможно возникновение ситуаций, когда может быть применено несколько правил. Отсюда следует

потребность в реализации некоторой управляющей структуры, которая будет определять очередность выполнения правил продукции.

Первоначально продукционные системы привлекли внимание разработчиков именно простотой механизма правил продукции. Однако в действующих экспертных системах, созданных на базе правил продукции, насчитывается до 1 000 правил и более. При таком количестве правил управление их выполнением превращается в серьезную проблему. Поэтому разработчики продукционных систем были вынуждены вводить в каноническую (принципиально простую) управляющую структуру те или иные усложнения, направленные на повышение эффективности. Одной из таких мер является введение нескольких уровней для правил, когда одни правила (метаправила) определяют порядок применения правил нижнего уровня.

Недостатком продукционных моделей считается отсутствие строгой теории и преобладание эвристики. Объяснение этой ситуации находят в расплывчатости понятия продукции и многообразии применяемых способов управления системами продукций. Представление проблемной области в виде множества продукций при их большом количестве влечет за собой проблему полноты и непротиворечивости.

В качестве еще одной проблемы продукционных систем называют переход от статических систем к динамическим, изменяющим в процессе своего функционирования состав системы продукций или модифицирующим алгоритмы разрешения конфликтных наборов на основе предыдущего опыта. Подобные системы продукций называют адаптивными, или системами продукций реального времени. Очевидно, что такие системы, построенные на принципах самообучения, представляют весьма перспективное направление.

6.13. Семантические модели

Семантикой называют смысл слов, текстов, действий или ситуаций и т. п., переданных с помощью некоторых представлений и выражений. Семантика определяет область общих отношений между символами и объектами, представленными этими символами; прагматика – отношения между символами и создателями этих символов, а синтактика – исключительно отношения между символами.

Семантические сети стали одним из основных способов представления знаний в технических системах имитации интеллекта [16]. Концепция семантической сети заимствована из психологии, где она использовалась в качестве модели человеческой памяти. В этой модели памяти, предложенной Куиллианом, для представления семантических отношений между концептами использовалась ассоциативная сетевая структура, вершины которой соответствуют концептам, а дуги – отношениям между концептами.

Пример ассоциативной структуры, называемой в модели Куиллиана *плоскостью*, представлен на рис. 6.12. В данной структуре для понятия моста отношение «класс» ассоциируется с понятием сооружения, свойства связаны с понятиями «деревянный», «железобетонный» и «стальной», с назначением – путь для преодоления некоторого препятствия (например, реки или оврага), а

примером может служить какой-либо стальной мост. Описываемые при этом концепты называются *вершинами типа*, а ассоциированные с ними слова – *вершинами лексем*. В каждой плоскости может быть только одна вершина типа и необходимое для его определения число вершин лексем. С помощью указателей может быть установлена связь между различными концептами.

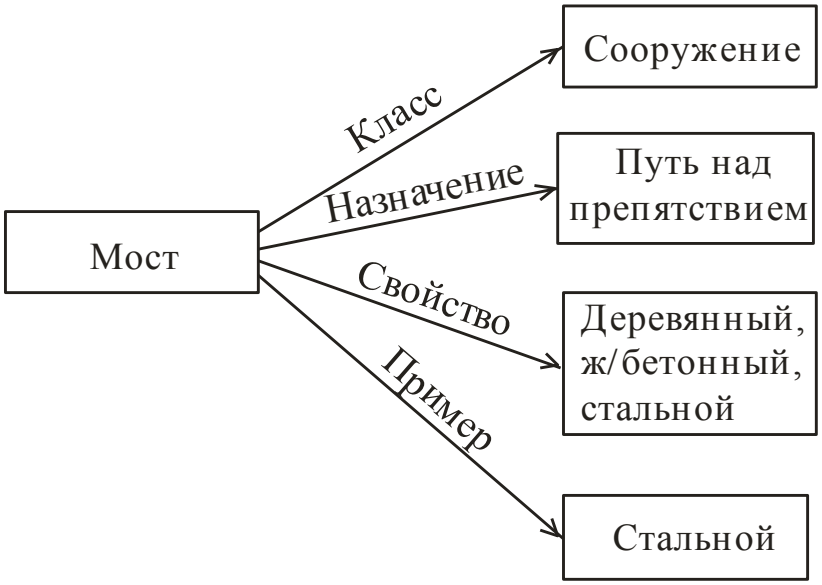


Рис. 6.12. К понятию семантической сети

Представление знаний семантическими моделями основано на идее, что память человека формируется через ассоциации между понятиями. И семантические модели знаний представляют собой воспроизведение в машине этого свойства памяти. Базовыми элементами семантических моделей знаний являются понятия и связи между ними. Поскольку каждое понятие, как правило, связано со многими другими, постольку получаемые при этом структуры образуют сети. По этой причине семантические модели называют также *семантическими сетями*, которые могут рассматриваться как помеченные ориентированные графы.

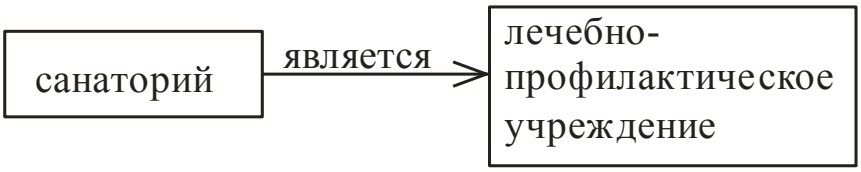
Вершины такого графа трактуются в самом широком смысле как сущности или понятия предметной области (объекты, события, процессы, явления, свойства и значения свойств), а *дуги* – как отношения между понятиями, которые они связывают. Любой из узлов графа может быть соединен дугами с любым числом других узлов. Направленность дуг отражает несимметричность моделируемых отношений и позволяет указывать роль понятий, например, выражать отношения «субъект – объект», «род – вид», «агрегат – компонент» и т. д.

В данной модели, в сущности, используется представление знаний в форме некоторых «элементов» и их «свойств». Следовательно, знания можно структурировать, заменив вершину типа на сущность (некоторый объект или элемент), а вершину лексемы – на свойство. Тогда для представления знаний в виде семантической сети потребуется всего три типа компонентов: сущности, свойства и указатели для представления отношений между сущностями и свойствами. *Сущностями* в семантической сети могут быть отдельные слова,

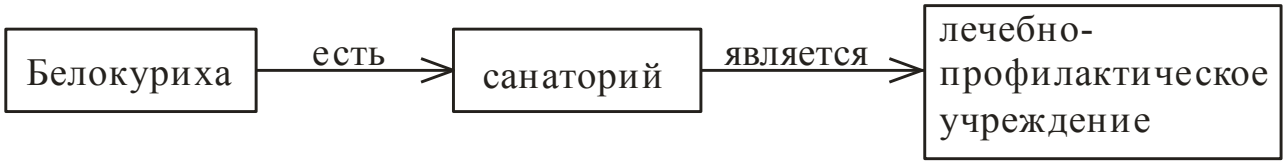
существительные, предложения или контекст. В сети сущности представляются как некоторые *факты*. *Свойства* – это структуры, описывающие сущности и соответствующие таким частям речи, как прилагательные, наречия, глаголы и т. п.

При построении семантической сети сущности, свойства и отношения желательно упорядочивать тем или иным образом. С этой целью используются такие отношения между понятиями, как подмножество – множество, логические связки $\vee$, $\wedge$, $\rightarrow$ и | (исключающее или) и другие. Кроме того, для выражения отношений между понятиями были предложены отношения «близости», «предпосылки», «сходства», «одновременности». Однако многие из этих отношений зависят от предметной области, и их унификация является очень сложной задачей. Поэтому в первую очередь интерес представляют отношения, имеющие место во многих мирах, то есть универсальные отношения.

Рассмотренная выше модель человеческой памяти в виде семантической сети демонстрирует связи между понятиями, но не раскрывает способов представления знаний. Поэтому ближайшей задачей является рассмотрение семантической сети как механизма представления и использования знаний. Возьмем в качестве примера тот факт, что любой санаторий является лечебно-профилактическим учреждением. Данный факт можно представить в виде простейшей семантической сети с двумя вершинами («санаторий» и «лечебно-профилактическое учреждение») и одной дугой («является»):



Если некоторый конкретный объект Белокуриха является санаторием, то мы можем дополнить нашу семантическую сеть этим фактом, в результате чего получим семантическую сеть:



Здесь следует обратить внимание на названия дуг. Вообще-то, с таким же успехом можно было использовать пару выражений «Белокуриха есть санаторий» и «санаторий есть лечебно-профилактическое учреждение» либо пару «Белокуриха является санаторием» и «санаторий является лечебно-профилактическим учреждением». Хотя в обоих случаях для обозначения отношений мы можем использовать одно и то же слово («есть» или «является»), эти два отношения представляют собой разные типы отношений.

Отношение между объектом, обозначенным как «Белокуриха», и объектом, названным «санаторий», есть отношение между эмпирическим объектом (конкретным санаторием) и абстрактным объектом – понятием санатория, которому соответствует множество объектов, попадающих под определение

санатория как такового, один из которых и есть санаторий Белокуриха. Таким образом, в данном случае мы имеем дело с отношением между множеством и элементом множества, которое называют также *отношением принадлежности*.

Отношение между понятиями «санаторий» и «лечебно-профилактическое учреждение» представляет собой отношение между двумя абстрактными понятиями, каждому из которых может быть поставлено в соответствие определенное множество индивидных объектов. При этом первое множество (санаториев) окажется подмножеством второго множества (лечебно-профилактических учреждений). Таким образом, данное отношение, во-первых, является отношением между двумя абстрактными объектами, и, во-вторых, отношением между множеством и подмножеством.

Поэтому, чтобы не вносить путаницу и различать эти два типа отношений, далее для обозначения отношения между эмпирическим объектом и соответствующим ему абстрактным объектом используется глагол «есть», а для выражения отношения между подмножеством и множеством – глагол «является».

В последнем примере представлены два факта, но на их основании можно вывести третий факт: «Белокуриха – лечебно-профилактическое учреждение». Данный пример показывает, что представление знаний в виде семантической сети позволяет относительно легко делать выводы, используя свойство наследования.

С помощью семантической сети можно представлять свойства объектов. Так, каждое лечебно-оздоровительное учреждение имеет медицинский персонал, что позволяет трансформировать нашу семантическую сеть и представить ее в следующем виде (рис. 6.13).

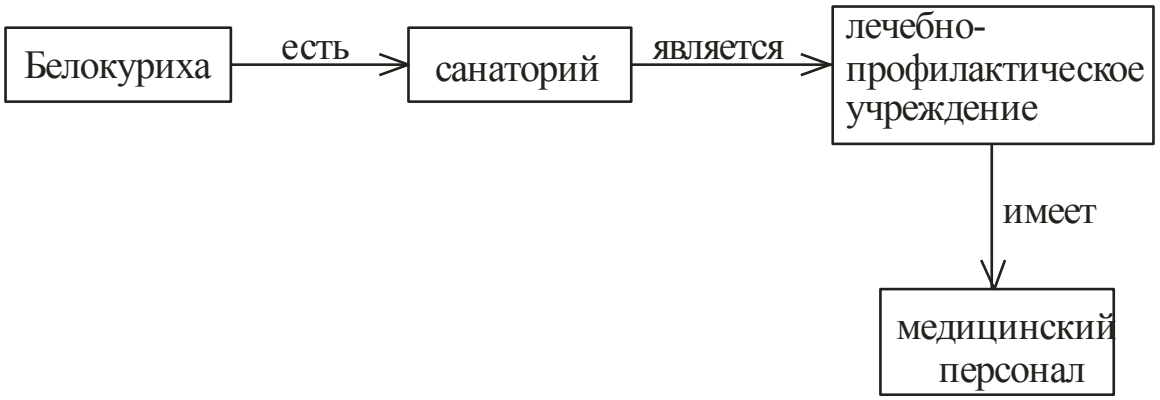


Рис. 6.13. К преобразованию семантической сети

Отсюда следует возможность вывода фактов, не представленных в сети в явном виде: каждый санаторий имеет медицинский персонал, в том числе и санаторий Белокуриха. Таким образом, любой факт, объявленный для вершин верхнего уровня иерархии, справедлив для вершин нижних уровней.

По мере расширения семантической сети в ней могут появляться новые типы сущностей, свойств и отношений. С принципиальной точки зрения подобные расширения семантической сети не представляют особых проблем. Если, например, требуется отразить тот факт, что санаторий Белокуриха владеет

определенным земельным участком (или участками), то семантическая сеть тогда примет вид, где земельный участок *i* есть конкретный земельный участок или экземпляр участка (рис. 6.14).

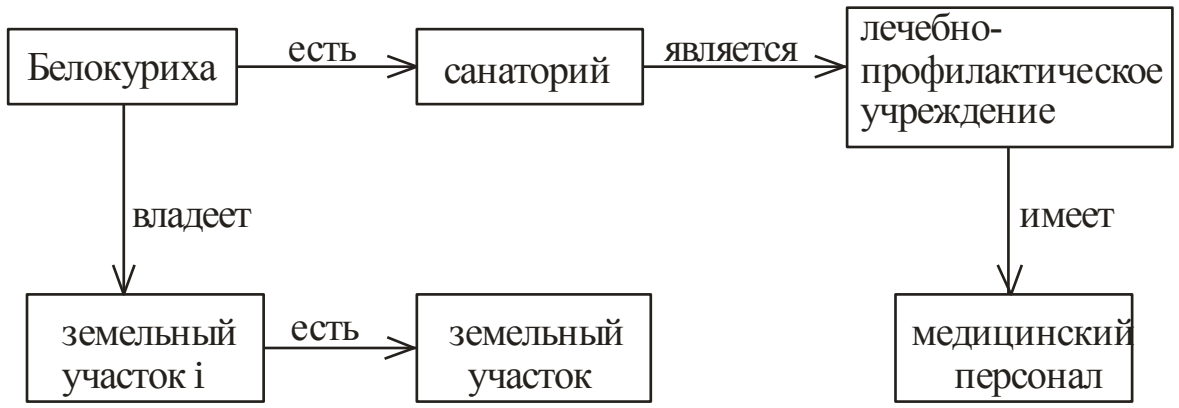


Рис. 6.14. Расширение семантической сети

Однако, при представлении знаний в виде семантических сетей возникают некоторые серьезные проблемы. Рассмотрим одну из них на том же примере. Допустим, что санатории проверяются некоторой комиссией, что можно представить так, как показано на рис. 6.15.

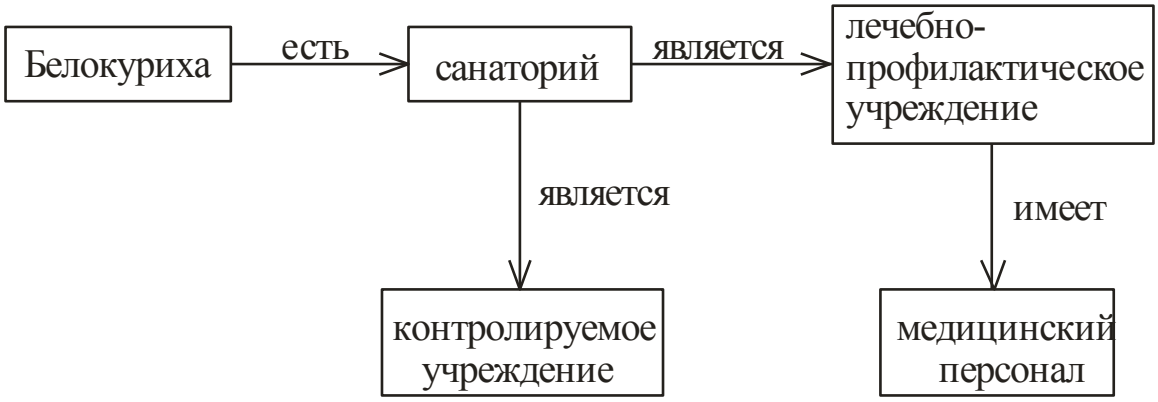


Рис. 6.15. К наследованию свойств

Из анализа полученной сети можно сделать вывод, что некоторые санатории, безусловно, проверялись комиссией. Но вывод о том, что санаторий Белокуриха также был подвергнут проверке, может оказаться неверным. Причина возможности получения неверного заключения состоит в том, что в семантической сети *процедура вывода* заменена *правилом наследования*. Данный пример показывает, что, во-первых, результат вывода, полученного с помощью семантической сети, не обладает достоверностью, присущей логическому формализму, и, во-вторых, требуется некоторый механизм управления наследованием свойств. Решение указанной проблемы основано на четком разделении вершин концептов (понятий, сущностей) и вершин экземпляров сущностей (индивидных объектов), то есть на разделении интенсиональных и экстенсиональных знаний.

Кроме указанной проблемы, в семантических сетях существует проблема представления процедурных знаний, в связи с чем были разработаны

процедурные семантические сети. Концепция процедурных семантических сетей подразумевает их построение на основе классов (понятий) и представление вершин, дуг (связей, отношений) и процедур как объектов.

Процедуры разделяются на два класса: процедуры для манипулирования вершинами и процедуры для управления связями. Процедуры для манипулирования вершинами выполняют такие функции, как:

- определение экземпляра класса;
- аннулирование экземпляров;
- подсчет числа экземпляров, принадлежащих к указанному классу;
- проверка принадлежности вершины некоторому классу.

Над дугами с помощью процедур осуществляются следующие основные действия:

- установление связи;
- аннулирование связи;
- определение числа вершин, соединяемых конкретной дугой;
- проверка существования связи определенного типа между заданными вершинами.

Использование указанных процедур дает возможность представлять операционные знания с помощью семантических сетей. Кроме того, в процедурных семантических сетях предпринимаются меры для управления наследованием. С этой целью атрибуты каждого класса разбиваются на *атрибуты определения* и *атрибуты свойства*. Атрибуты свойства представляются как отношения между классами и не наследуются нижними классами иерархии. Кроме того, для управления наследованием между двумя уровнями может использоваться индивидуальное наследование, когда процедура наследования состоит в том, что наследование атрибута осуществляется с помощью отношения «есть» (между концептом и экземпляром), а наследование значений – с помощью отношения «является» (отношение таксономии). Наследование значений атрибутов свойств не производится. (Следует заметить разницу в толковании понятий «атрибут» и «свойство». Обычно эти термины рассматриваются почти как синонимы. Здесь же атрибутами могут быть разные понятия или концепты, в том числе и свойства. Таким образом, в данном контексте понятие атрибута по смыслу близко к понятию функционального места или роли.) Поэтому в тех случаях, когда требуется, чтобы какой-либо концепт (вершина класса *x*) наследовал специфические атрибуты класса верхнего уровня, этот класс верхнего уровня снабжается атрибутом *метакласса*, в котором класс *x* является экземпляром.

Особенность систем, созданных на основе семантической сети, заключается в их целостности. Эта целостность одновременно является и их недостатком, поскольку не позволяет разделить базу знаний и механизм выводов.

Интерпретация семантической сети осуществляется использующими ее процедурами, основанными на нескольких способах. Наиболее типичным таким способом является *способ сопоставления частей сетевой структуры*,

суть которого состоит в построении подсети, соответствующей запросу, и ее сопоставлении с семантической сетью.

Рассмотрим такое сопоставление на следующем примере. Пусть существует семантическая сеть, фрагмент которой приведен на рис. 6.16, и требуется выяснить, что имеет Белокуриха. В соответствии с этим запросом системой будет создана подсеть, изображенная на рис. 6.17, и выполнено сопоставление с семантической сетью. Поиск начинается с вершины «имеет», у которой дуга «субъект» (кто имеет) направлена к вершине «Белокуриха». Когда такая вершина найдена, осуществляется следование по дуге «объект» (что имеет) к соответствующей вершине и выбирается ее значение, которое и есть ответ на поставленный вопрос.

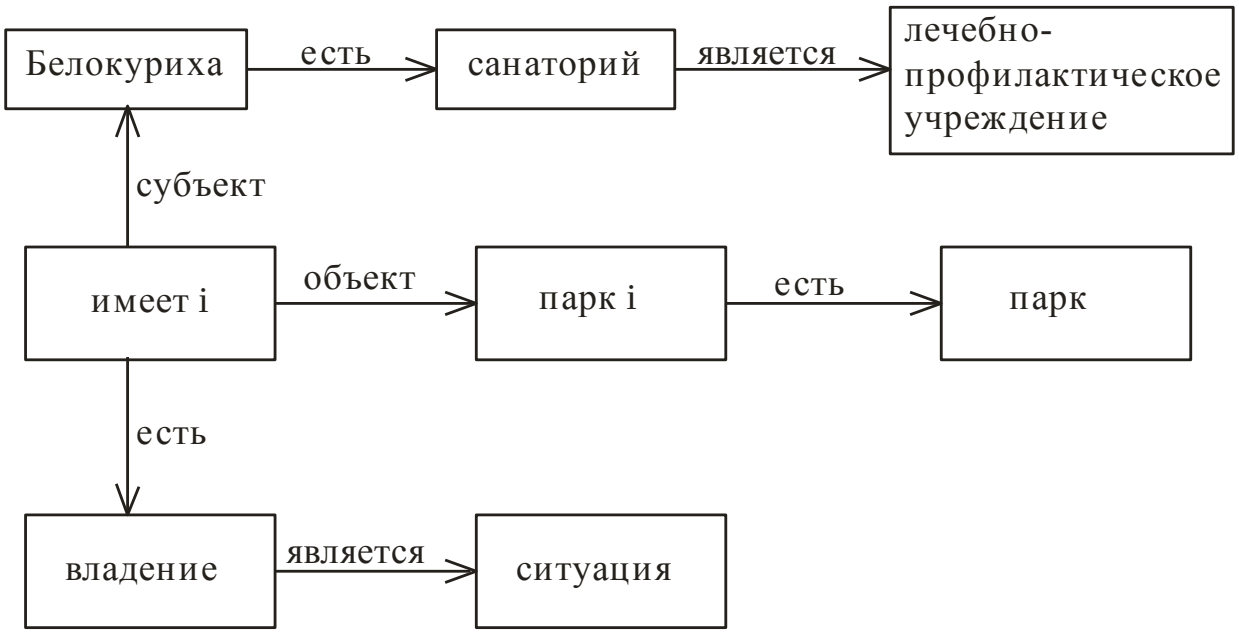


Рис. 6.16. Фрагмент семантической сети

Пусть теперь требуется ответить на вопрос: «Существует ли лечебно-профилактическое учреждение, имеющее парк?» Данный вопрос можно представить в виде подсети, изображенной на рис. 6.18. Но данная подсеть не пригодна для непосредственного сопоставления с семантической сетью. Чтобы сравнение стало возможным, от вершины «Белокуриха» необходимо провести дугу «есть» к вершине «лечебно-профилактическое учреждение». Смысл этого действия в том, что таким образом выражается факт «Белокуриха есть лечебно-профилактическое учреждение». Теперь становится возможным соединение вершины «имеет?» с вершиной «имеет *i*» и вершины «парк?» с вершиной «парк *i*». Только после этого можно выполнить сопоставление и получить утвердительный ответ: «Да, это Белокуриха».

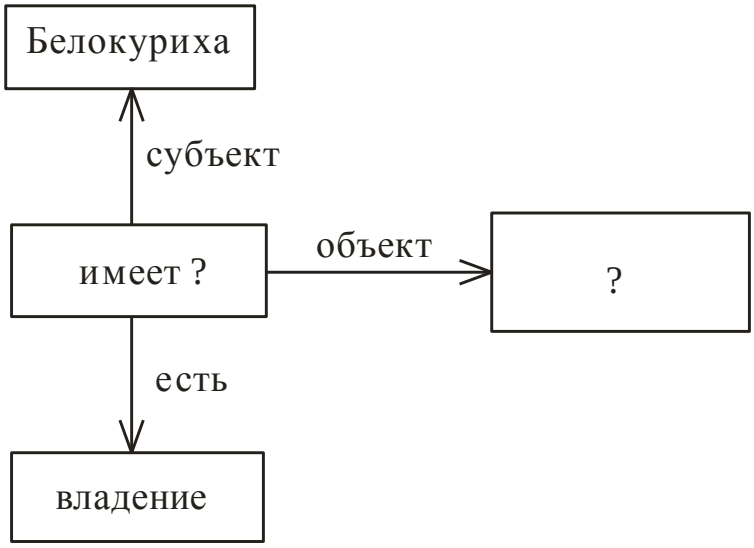


Рис. 6.17. Подграф запроса 1

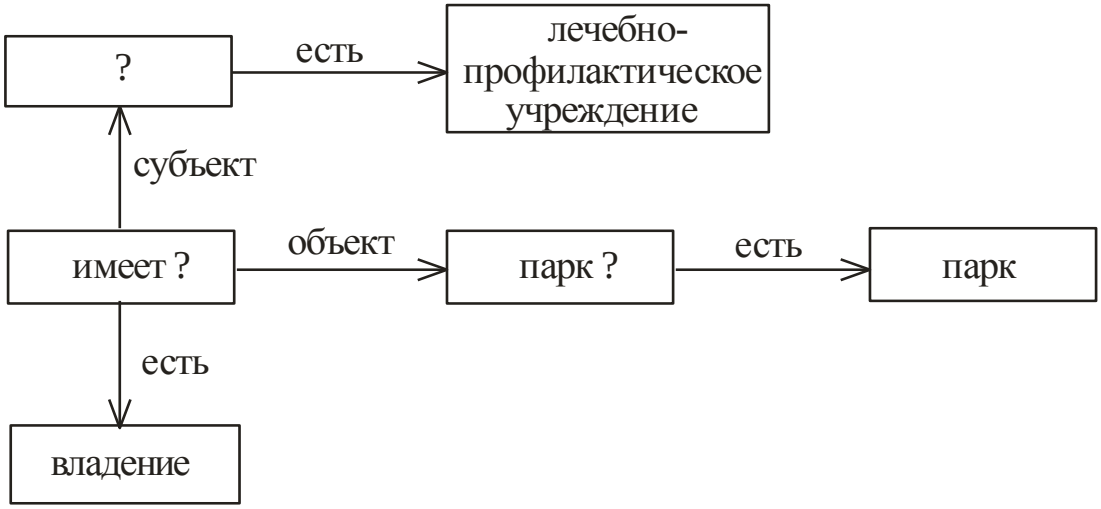


Рис. 6.18. Подсеть запроса 2

Формализация семантических знаний о предметной области осуществляется на основе абстрактного ориентированного графа с помеченными вершинами и дугами, называемого *семантической сетью* [13]. Вершины такого графа отождествляются с некоторыми объектами, а дуги – с отношениями между ними. Метки вершин служат для их идентификации и представляют собой слова естественного языка либо другие символы, смысл которых вполне определен. Метки дуг обозначают конкретные отношения.

Формальное определение семантической сети сводится к следующему. Пусть дано некоторое число конечных множеств символов $A = \{A_1, ..., A_n\}$, называемых атрибутами, и конечное множество отношений $R = \{R_1, ..., R_m\}$. Тогда *схемой,* или *интенсионалом отношения* R_i, называют набор пар:

$$INT(R_i) = \{..., [A_j, DOM(A_j)], ...\},$$

где R_i – имя отношения; $DOM(A_j)$ – множество допустимых значений атрибута A_j отношения R_i или *домен* A_j. Объединение всех доменов

представляет собой *множество объектов*, на которых задаются отношения R_i, и называется *базовым множеством семантической модели*.

Экстенсионалом отношения R_i называют множество

$EXT(R_i) = \{F_1, \ldots, F_j\}$,

где F_k $(k = 1, ..., l)$ – так называемые *факты отношения* R_i. Каждый факт задается *атрибутивной парой* вида «атрибут – значение» и представляет собой конкретизацию выделенного отношения между указанными объектами.

В графическом представлении факт – это звездный подграф семантической сети, корень которого является вершиной предикатного типа с уникальной меткой, включающей имя соответствующего отношения. Дуги помечаются именами атрибутов данного факта и соединяют вершину факта с вершинами базового множества, являющимися значениями этих атрибутов.

Отношение между двумя понятиями может рассматриваться как простой факт. С логической точки зрения две вершины и связывающая их дуга могут рассматриваться как предикат с двумя аргументами (бинарный предикат). При этом оба аргумента представляются двумя вершинами, а сам предикат – помеченной дугой, связывающей эти вершины. Тогда каждая дуга может трактоваться как утверждение о некотором простом факте, а вся сеть – как множество фиксированных утверждений о предметной области.

Рассмотрим пример определения семантической сети. Пусть имеется множество городов {А, Б, В, Г}, множество отношений, состоящее из отношений двух видов: «численность больше – численность меньше», которое обозначим символом «>», и отношение «автобусное сообщение», которое обозначим символом «а», с указанием длины автобусного маршрута.

Тогда интенсионалы этих отношений можно формально представить в следующем виде:

INT (>) = {[меньше, (А, Б, В)], [больше, (Б, В, Г)]};

INT (а) = {[город1, (А, Б, В, Г)], [город2, (А, Б, В, Г)], [длина маршрута, (90, 120, 140, 150]}.

Пусть экстенсионалы данных отношений представляют собой следующие факты, которые мы запишем как:

F_1: (автобусное сообщение, город1 А, город2 Б);

…

F_5: (автобусное сообщение, город1 В, город2 Г);

F_6: (>, больше А, меньше Б);

…

F_{10}: (>, больше В, меньше Г).

В графическом виде экстенсиональная сеть представлена на рис. 6.19, где:

– элементы базового множества, включающего два домена, обозначены квадратами;

– вершины фактов обозначены окружностями, метки которых содержат идентификатор факта и тип отношения, описываемого соответствующим фактом;

– для обозначения меток дуг, именующих атрибуты отношений, приняты следующие сокращения: «город1» – «г1», «город2» – «г2», «длина (маршрута)» – «д», «больше» – «б» и «меньше» – «м».

На данном примере можно убедиться в том, что графический способ изображения не пригоден для представления реальных семантических сетей, и его применение преимущественно сводится к демонстрации основных понятий и принципов организации семантических сетей. Для представления экстенсиональных семантических сетей более удобным является табличный способ представления (табл. 6.1, 6.2). Преимущество графического способа перед табличным проявляется только в том, что семантическая сеть при его использовании предстает в виде целостного образования, системы. Применение же табличного способа не дает ощущения целостности семантической сети.

Экстенсиональная семантическая сеть содержит экстенсиональные знания об объектах предметной области и отражает ее состояние на определенный момент времени. Интенсиональная семантическая сеть содержит интенсиональные знания о предметной области в виде ее общей структуры как отношений между абстрактными объектами.

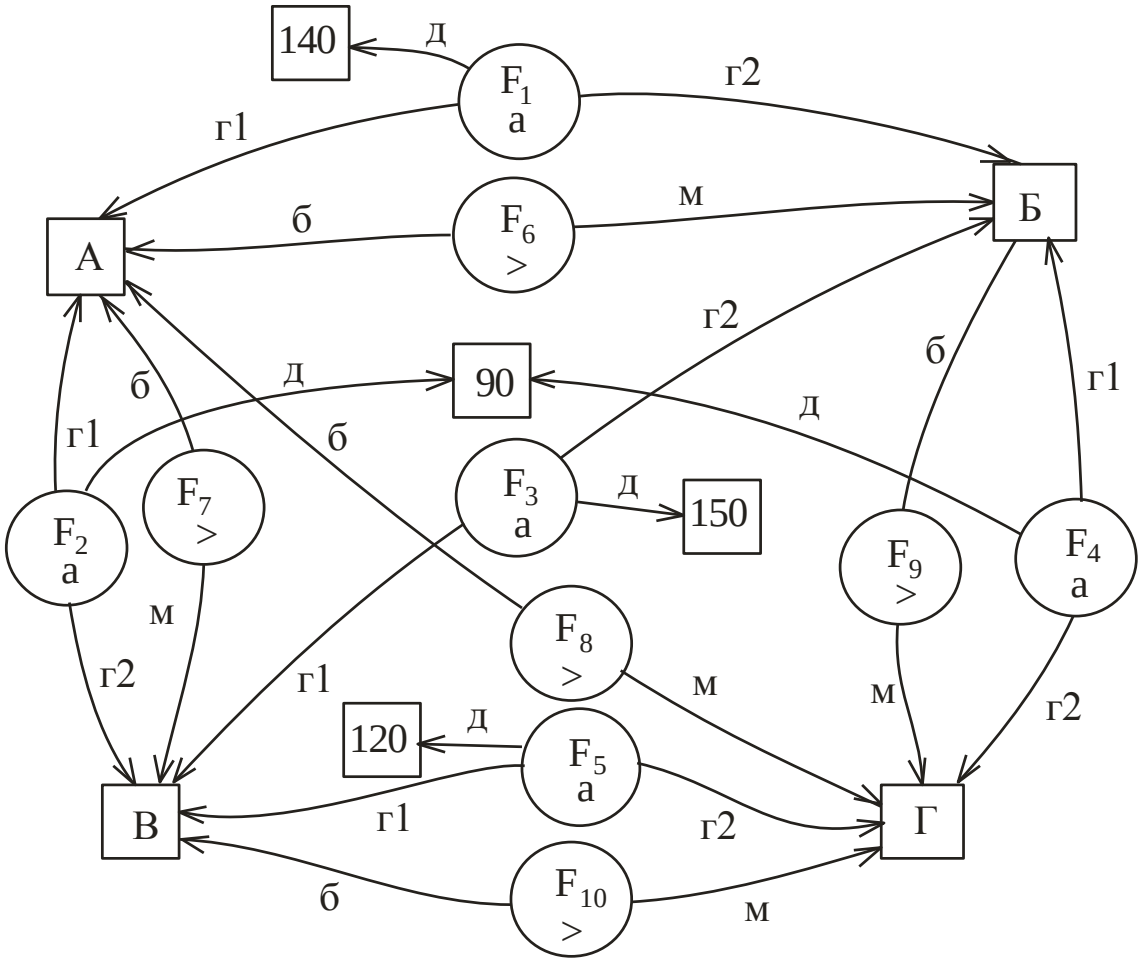


Рис. 6.19. Экстенсиональная семантическая сеть

Таблица 6.1. Отношение «автобусное сообщение»

Метка факта	Город1	Город2	Длина маршрута
F_1	А	Б	140
F_2	А	В	90
F_3	В	Б	150
F_4	Б	Г	90
F_5	В	Г	120

Таблица 6.2. Отношение «больше – меньше»

Метка факта	больше	меньше
F_6	А	Б
F_7	А	В
F_8	А	Г
F_9	Б	Г
F_{10}	В	Г

В семантических сетях иногда различают три типа элементов: понятия, события и свойства. *Понятия* трактуются как сведения об абстрактных и индивидных объектах предметной области, а общие понятия – как множества доменов. По сути, здесь под общими понятиями подразумеваются типы объектов предметной области.

События рассматриваются как действия, в результате которых происходят изменения в предметной области. Результатом события является новое состояние предметной области. Из такой трактовки следует, что приведение предметной области из исходного состояния к некоторому желательному состоянию составляет последовательность промежуточных состояний (траектория предметной области), являющихся результатом определенных событий. Тогда решение задачи преобразования предметной области к целевому состоянию сводится к поиску в семантической сети необходимой последовательности событий (действий, операторов).

Свойства в семантических сетях используются для уточнения понятий, событий и других свойств. Использование свойств для описания понятий заключается в определении их особенностей или характеристик, параметров (состояние, назначение, материал и т. д.). Свойства событий – это продолжительность, время и т. п.

Всю совокупность семантических отношений иногда подразделяют на четыре класса: лингвистические, логические, теоретико-множественные и квантифицированные отношения.

Лингвистические отношения подразделяются на следующие:

1. *падежные* (наиболее употребительные), среди которых обычно выделяют:

– *агент* – отношение между событием и тем, кто или что его вызывает;

– *объект* – отношение между событием и тем, на что направлено действие;

– *условие* – отношение, указывающее на логическую связь между событиями;

– *инструмент* – отношение между событием и объектом, с помощью которого оно совершается;

– *место* – место совершения события;

2. *характеризацию глаголов*, к которой относят наклонение, время, род, число, залог;

3. *атрибутивные*, под которыми понимают различные свойства объектов (форма, состояние, особенности конструкции и т. п.).

Логические отношения – это отношения, используемые в исчислении высказываний: *отрицание*, *дизъюнкция*, *конъюнкция* и *импликация*.

Теоретико-множественные отношения включают отношение *подмножества*, обозначаемое SUB, *супермножества*, обозначаемое SUP, элемент множества ($\in$), отношение «часть – целое» и т. п. С помощью данного класса отношений строится таксономия объектов, обладающая тем важным свойством, что свойства SUP-сущности наследуются SUB-сущностями, что позволяет сокращать объем баз знаний.

Квантифицированные отношения служат общим названием для логических кванторов существования и общности. В принципе, их можно было не выделять в качестве самостоятельного класса, а отнести к логическим отношениям. Квантифицированные отношения обычно используются для представления декларативных знаний, например, таких: «Все шлюзы имеют хотя бы одни ворота», «Все нефтепроводы являются трубопроводами», «Некоторые лесные пожары являются верховыми» или «Некоторые полагают, что внимание дороже подарка».

Особое место среди разнообразных отношений между моделируемыми объектами занимают отношение принадлежности множеству («множество – элемент») и отношение таксономии («род – вид»). Если есть множество индивидных объектов некоторого типа, то нет необходимости для каждого эмпирического объекта указывать свойства, присущие классу таких объектов в целом, а вполне достаточно указать их один раз для всего класса.

Так, если в некоторой геоинформационной системе хранятся модели некоторого числа рек, то для каждой конкретной реки излишне указывать в базе данных, что она может

– использоваться как источник водоснабжения;

– служить водным путем;

– являться препятствием для наземных видов транспорта;

– служить средой для размножения некоторых видов болезнетворных бактерий;

– служить препятствием для распространения пожаров;

– служить административной или государственной границей;

– обладать некоторым запасом биологических ресурсов,

а также другими свойствами, которые могут представлять тот или иной интерес. Вместо этого все перечисленные свойства следует представить один раз в базе знаний как характеристики класса «река».

Семантические сети являются удобным средством для представления отношения таксономии вследствие возможности «*наследования*» свойств, в результате которого абстрактным объектам нижнего уровня иерархии приписываются значения всех свойств, характеризующих объекты верхнего уровня.

Важной особенностью семантических сетей является возможность манипулирования некоторыми сущностями при отсутствии полной информации о них. С этой целью вводится уникальное внутрисистемное имя, называемое *десигнатом*. (Заметим, что термин «десигнат» обычно употребляется как синоним понятия «обозначаемое», здесь же, как видим, он используется для указания на обозначающее.) Основной функцией десигната является представление основных свойств некоторого объекта, иногда – отражение самого факта существования такого объекта. С течением времени свойства такого объекта могут выясняться и заноситься в базу данных.

Таким образом, десигнат может быть создан в момент первого появления объекта в системе, и им можно манипулировать как объектом, соблюдая некоторые ограничения. Использование десигнатов, безусловно, является полезным приемом, поскольку придает семантической сети определенную гибкость.

Основным принципом создания интеллектуальных систем на основе семантических сетей является разделение интенсиональных и экстенсиональных знаний. При таком разделении интенсиональные знания представляют собственно базу знаний, а экстенсиональные знания есть не что иное, как база данных о предметной области. В литературе отмечается, что раздельное представление интенсиональных и экстенсиональных знаний позволяет повысить выразительные возможности системы по сравнению с другими моделями, например, логическими.

Кроме того, необходимо отметить возможность повышения достоверности фактов, хранимых в базе данных, поскольку для проверки многих значений или соотношений атрибутов могут использоваться интенсиональные знания. Примеры использования подобных знаний:

- отметки урезов не должны возрастать вниз по течению водотоков;
- высоты верхней бровки откосов должны быть больше высот по низу откоса;
- глубина брода, высота и диаметр деревьев, грузоподъемность мостов … не должны превышать соответствующих предельных величин и т. п.

Разделение экстенсиональных и интенсиональных знаний влечет за собой двухуровневую организацию интеллектуальных систем, создаваемых на основе семантических сетей (рис. 6.20). В системах геомоделирования объем базы данных может превышать объем базы знаний на 3–4 порядка или более.

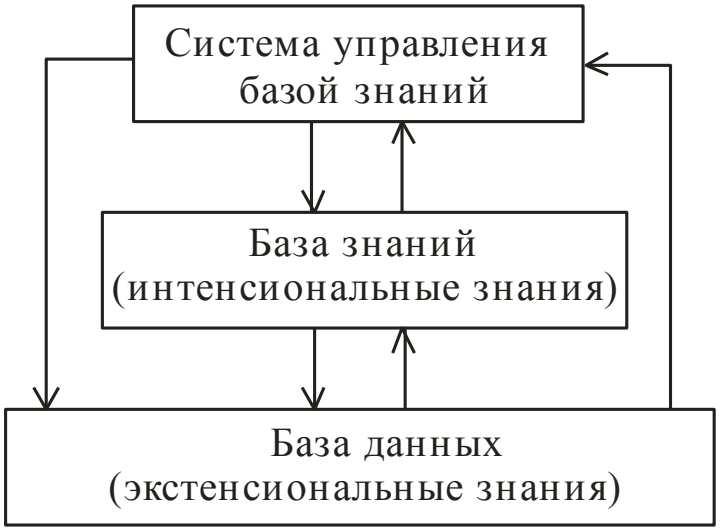


Рис. 6.20. Структура представления знания

В процессе функционирования системы представления знаний возможны два режима: ведение базы знаний (режим обучения) и решение тех или иных задач. Последний режим является основным. Использование базы интенсиональных знаний заключается преимущественно в выполнении информационно-поисковых задач. Каждый запрос к базе знаний содержит совокупность фактов, описывающих ситуацию. При этом в позициях значений атрибутов, имен атрибутов и имен отношений допускается использование переменных. Запрос может быть представлен графом, вершины которого, соответствующие некоторым переменным, не определены. Поиск ответа сводится к решению задачи изоморфного вложения графа запроса или его подграфа в семантическую сеть, как это было продемонстрировано в примере с Белокурихой.

Различают два основных типа запросов: на существование и на перечисление. Запрос на существование состоит в проверке наличия некоторого факта в базе знаний и не содержит переменных. Ответом на подобные запросы является либо ДА, если изоморфное вложение графа запроса в семантическую сеть удалось, либо НЕТ – в противном случае.

При выполнении запроса на перечисление осуществляется поиск всех подграфов семантической сети, изоморфных графу запроса, и конкретизация переменных, то есть присваивание переменным значений соответствующих констант.

Семантические сети позволяют представлять и декларативные, и процедурные знания. Декларативные знания представляются с использованием так называемых *базовых отношений*, особенность которых заключается в том, что их экстенсионалы полностью хранятся в базе данных. Процедурные знания основаны на виртуальных отношениях, не имеющих хранимых в явной форме экстенсионалов. Описания таких отношений содержат правила, позволяющие выводить интересующие факты из имеющихся. Виртуальные отношения реализуются:

– в виде вычислимых отношений, имеющих функциональный или предикатный характер;

– в виде правил, использующих свойства рефлексивности, симметричности и транзитивности отношений, присущих предметной области.

Семантические модели обеспечивают возможность представления отношений между индивидными объектами, между абстрактными, то есть общими понятиями, а также между индивидными объектами, с одной стороны, и абстрактными – с другой. Так, с помощью семантической сети можно представить утверждения «Реки являются объектами гидрографии», «Уссури является рекой», «Уссури является притоком Амура» (рис. 6.21). В первом утверждении денотатами обоих терминов являются абстрактные объекты. Во втором утверждении устанавливается связь между индивидным и абстрактным объектами. Последнее утверждение описывает отношение между двумя эмпирическими объектами.

Определяя типы вершин и дуг, можно получить сети различного вида. Если вершины сети не имеют собственной внутренней структуры, то такую сеть называют *простой*. Если хотя бы некоторые вершины сети обладают некоторой структурой, то такую сеть называют *иерархической*. На начальном этапе использования семантических моделей использовались только простые сети, но с течением времени все чаще стали применяться иерархические сети. Фрагмент гипотетической семантической сети приведен на рис. 6.22.

Рис. 6.21. Пример 1 семантической сети

Как следует из рис. 6.22, все множество понятий разделяется на *объекты*, *свойства* и *отношения*. В свою очередь, в свойствах различаются *наименования свойств* и *значения свойств*. Кроме того, свойства подразделяются на *количественные* и *качественные*. С количественными свойствами связаны *единицы измерений*.

В рассматриваемом примере разновидностями типов объектов являются класс «здания и постройки» и класс «дороги». Таким образом, дуга между

вершиной «объекты» и вершиной «дороги», а также дуга «объекты» – «здания и постройки» выражают отношение таксономии или родовидовое отношение, существующее между этими понятиями. Между понятием «здание», с одной стороны, и понятиями «коробка (здания)» и «крыльцо», с другой стороны, существует отношение агрегации, в котором здание играет роль агрегата (составного объекта), а коробка и крыльцо являются его компонентами.

Дуга между понятием «отношение» и понятием «отношение агрегации» является изображением отношения между множеством (понятием отношения) и его элементом (отношением агрегации).

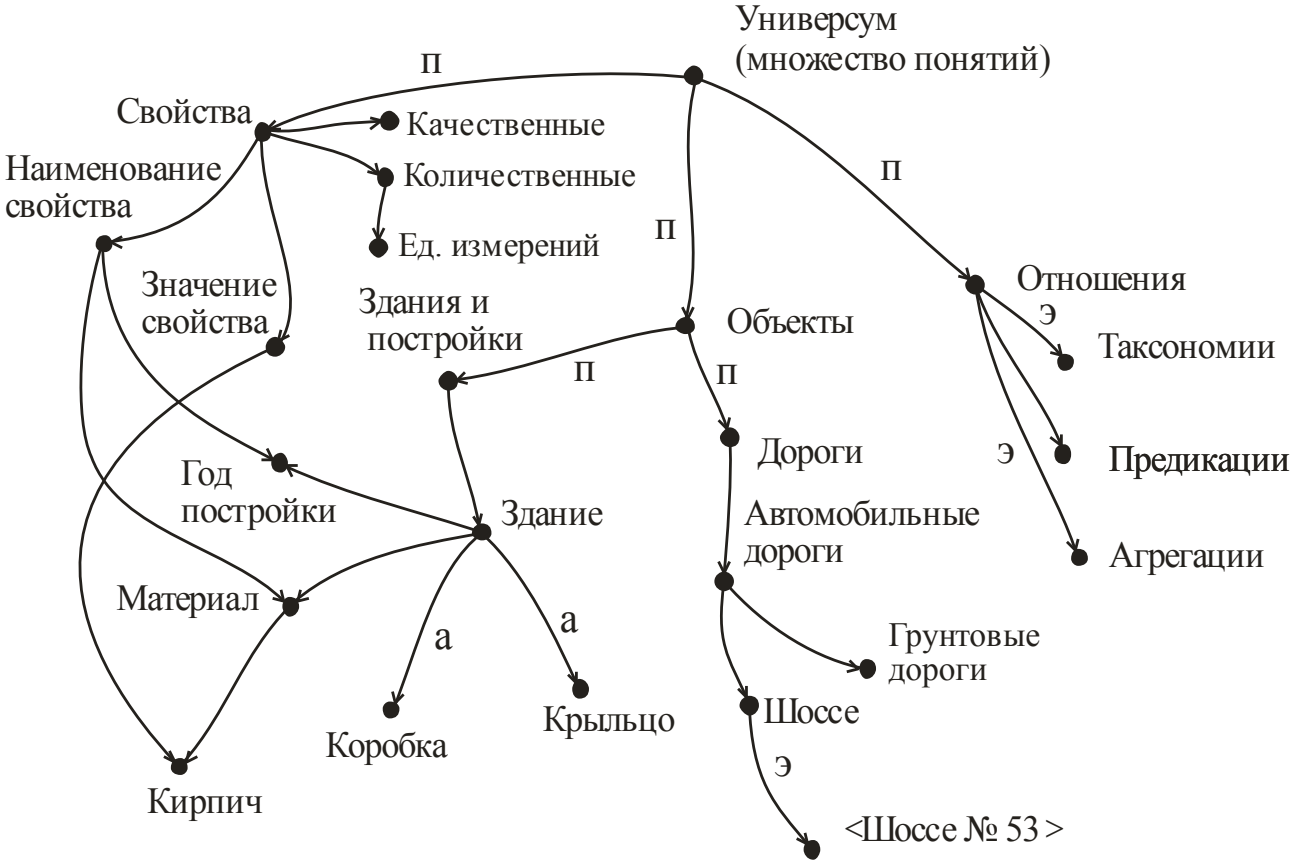


Рис. 6.22. Пример 2 семантической сети

Упомянутые выше отношения таксономии, агрегации, а также отношение предикации, отношение между множеством и его элементами, между множеством и подмножеством при моделировании внешнего мира передают наиболее важные виды связей между понятиями и встречаются наиболее часто. По этим причинам их иногда называют *фундаментальными* отношениями. Отношение таксономии играет важную роль в связи с тем, что объект нижнего уровня иерархии наследует все свойства объекта верхнего уровня.

На рис. 6.22 некоторые дуги помечены; при этом буква «э» обозначает отношение между элементом множества и множеством, буква «п» – отношение между подмножеством и множеством, буква «а» – отношение между агрегатом и компонентом. Следует обратить внимание на то, что вершины интенсиональной семантической сети, как правило, соответствуют не отдельным сущностям, а множествам, обозначаемым с помощью вершин.

Наряду с интенсиональными знаниями в семантической сети, в данном примере представлены и экстенсионалы. Так, дуга между вершиной «шоссе» и вершиной «шоссе № 53» выражает отношение именования между символом («шоссе») и индивидным объектом. При использовании нотации Нормана – Румельхарта имена эмпирических объектов заключаются в угловые скобки, а имена абстрактных объектов (понятия) записываются без скобок.

Большинство подмножеств часто являются различными, то есть непересекающимися, поэтому иногда для их обозначения вводится дуга специального вида «пр» (подмножество различное).

Основным отличием иерархических семантических сетей от простых сетей является возможность разделения сети на некоторое число подсетей – *пространств* – и установления связей не только между вершинами, но и между пространствами. Каждая вершина и каждая дуга принадлежат одному пространству. Пространства при этом играют такую же роль, как скобки в математической нотации.

Все пространства сети упорядочиваются в виде *дерева пространств*. Вершинами дерева пространств являются пространства, а дуги соответствуют *отношениям видимости.* Отношение видимости означает, что вершины и дуги одного пространства доступны из другого пространства. Из пространства – потомка видимы все вершины и все дуги родительского пространства и пространств, по отношению к которым родительское пространство является потомком. Отношение видимости дает возможность сгруппировать пространства в упорядоченные множества, называемые *перспективами*. Обычно в перспективу включаются не произвольные, а только видимые пространства, что позволяет упорядочить пространства. Такое упорядочивание позволяет процедурам, работающим с пространствами, повысить эффективность поиска знаний, так как невидимые в каждый момент времени пространства исключаются из области поиска.

Из представления знаний в виде графа следует некоторая ограниченность представления знаний в виде семантической сети. Эта ограниченность вытекает из того, что в графе все отношения являются бинарными. Ранее, при обсуждении реляционных баз данных уже отмечалось, что использование только бинарных отношений не позволяет отобразить все многообразие свойств и отношений реального мира, и поэтому требуется представлять тернарные отношения и отношения более высокого порядка.

В заключение можно отметить основные свойства представления знаний с помощью семантических сетей. Положительной стороной данного способа является простота и понятность описания посредством отношений между узлами и дугами. Его отрицательные стороны:

- возрастание времени обработки при увеличении размеров сети;
- проблема справедливости полученных выводов, в том числе проблема наследования свойств.

6.14. Фреймовые модели

Автором фреймовой модели является американский математик М. Минский. Основными свойствами фреймовых (от англ. frame – рамка) моделей являются:

- представление знаний в виде крупных блоков, называемых *фреймами*;
- иерархическое построение системы фреймов, основанное на степени абстракции;
- сочетание декларативных и процедурных знаний.

Представление знаний фреймами так же, как и использование семантических сетей, базируется на использовании некоторых концепций из области психологии, используемых для объяснения восприятия и понимания человеком тех или иных сцен (ситуаций). (Интенсивное заимствование результатов исследований человеческого мышления и их перенос в экспертные системы стали основой для некоторых утверждений о том, что искусственный интеллект – это область знаний, лежащая между психологией и вычислительной техникой [21].)

Понимание человеком полученной информации происходит благодаря способности мозга увязывать ее с особыми концептуальными объектами, хранящимися в памяти. Теория фреймов объясняет восприятие фактов человеком через сопоставление полученной информации с некоторыми элементами и «рамками», хранящимися в памяти для каждого концептуального объекта. Структура таких рамок называется *фреймом*. Между различными концептуальными объектами имеется определенное сходство, поэтому их совокупность может быть представлена в виде иерархической системы с некоторыми отношениями. Среди таких отношений важную роль играют отношение «абстрактное – конкретное» и отношение обобщения. Сложные объекты представляются комбинациями нескольких фреймов, в результате чего фреймовая модель представления знаний образует сетевую структуру.

Для представления структуры общих понятий выбранной предметной области фреймы с помощью указателей, в качестве которых используются имена фреймов, объединяются в сети. Фреймы используются не только для описания общих понятий, но и для представления индивидных объектов.

Кроме того, с каждым фреймом связываются релевантные факты и процедуры, осуществляющие запросы к другим фреймам. Такая дополнительная информация является специфической для каждого фрейма и может содержать способ обращения с данным фреймом, последовательность действий, указания о действиях, если некоторое предположение не оправдалось, и т. п.

Фрейм имеет *имя* для идентификации описываемого им понятия. Это описание состоит из ряда именуемых описаний, называемых *слотами*. С помощью слотов идентифицируются основные структурные элементы системы понятий и индивидных объектов. С каждым слотом связаны *шпации*, в которые помещаются некоторые объекты, представляющие значения слотов. В качестве значений слотов могут выступать любые объекты данных. В частности, ими могут быть целые и вещественные числа, символы и строки, структуры данных,

имена переменных и функций, имена процедур и указатели, в том числе, на другие фреймы. В любой слот могут помещаться значения не произвольные, а вполне определенного типа, называемого *фасетом*.

Фрейм может рассматриваться как определенная структура данных, а фреймовая система – как сетевая структура, узлы которой представляют собой фреймы. Для построения фреймовой системы используются следующие элементы: имя фрейма, слот, значение слота, указатель, тип данных и процедура.

Именем фрейма называют идентификатор, присваиваемый фрейму. Имя фрейма должно быть уникальным в конкретной фреймовой системе. Каждый фрейм состоит из некоторого числа слотов, часть которых автоматически определяется самой системой, а часть – определяется пользователем. Каждый слот создается с определенной целью и/или для выполнения некоторых функций и имеет ту или иную структуру, которая зависит от назначения слота. Таким образом, слот – это также некоторая структура данных. Основными слотами являются слоты для представления отношений между понятиями. В их число обычно входят слоты, указывающие родовой фрейм для каждого конкретного фрейма, и слоты для указания на фреймы – разновидности данного фрейма. Ссылки на эти и другие фреймы осуществляются с помощью указателей.

Именем слота называют присваиваемый слоту идентификатор, который должен быть уникальным во включающем фрейме. В принципе, имена слотов могут быть произвольными, то есть не нести никакой смысловой нагрузки. Но для некоторых слотов необходимо использовать фиксированные имена, имеющие один и тот же смысл в любом фрейме. К таким именам относятся имена отношения таксономии (IS-A), отношения агрегации (PART-OF), имена указателей, имена комментариев (COMMENT) и ряд других. Слоты с подобными именами являются *системными* (служебными) и используются при редактировании знаний и управлении выводом.

Особое место во фреймовых системах, основанных на отношении «абстрактное – конкретное» занимают *указатели наследования*. Они используются для сообщения системе информации о том, значения каких слотов во фреймах верхнего уровня могут быть присвоены значениям слотов во фреймах нижнего уровня. С их помощью можно указать, что в каждом фрейме выделенный слот может иметь произвольное значение, либо что

- все слоты данного типа должны иметь одинаковые значения,
- значения слотов во фреймах нижнего уровня должны находиться в пределах, заданных в слоте фрейма верхнего уровня,
- при отсутствии указания значения слота во фрейме нижнего уровня в качестве такового принимается значение слота из фрейма верхнего уровня, то есть значение по умолчанию, которое позднее может быть заменено фактическим значением.

В одних системах при этом допускается возможность одновременного существования разных вариантов наследования, а в других может быть лишь один из перечисленных вариантов.

С каждым слотом связан *тип данных*, указывающий на то, что все значения слота должны быть только одного указанного для него типа: INTEGER (целое), REAL (вещественное число), BOOL (истинностное значение), CHAR (символ), TEXT (текст), FRAME (указатель на фрейм), TABLE (таблица), PROC (процедура), EXPRESSION (выражение) и другие. Каждое вводимое значение слота должно быть допустимого типа и отвечать ограничениям, если они указаны в слоте фрейма верхнего уровня.

Особенностью фреймовых систем является то, что значениями слотов могут быть процедуры (программы), называемые *присоединенными*. Присоединенные процедуры могут запускаться после получения сообщения из другого фрейма. Таким образом, фреймовые модели объединяют декларативные знания и процедуры, то есть операционные знания. Поскольку во фреймовых моделях отсутствует специальный механизм управления выводом, постольку такой вывод должен быть реализован с помощью присоединенных процедур.

Для управления выводом во фреймовых системах используются три основных способа: с помощью процедур-демонов, с помощью служебных процедур и с применением механизма наследования. Последний способ, в сущности, является основным механизмом управления выводом во фреймовых системах. Благодаря этому механизму, основанному на отношении «абстрактное – конкретное», то есть на отношении типа «есть», выполняется автоматический поиск и определение значений слотов во фреймах верхнего уровня и служебных присоединенных процедур.

При организации фреймов стараются не дублировать одну и ту же информацию во всех фреймах, принадлежащих одной ветви, и она хранится, как правило, только в соответствующем единственном верхнем фрейме, вследствие чего удается сократить объем требуемой памяти. Поэтому механизм наследования имеет важное значение при последовательном управлении системами с базами знаний. С помощью процедур-демонов и служебных процедур в принципе можно реализовать любой механизм управления выводом. Тем не менее, считается, что для корректного функционирования всей системы в целом чрезвычайно важно ее тщательное проектирование и разработка достаточного количества присоединенных процедур. Наличие таких процедур принципиально отличает интеллектуальные системы от обычных программ обработки данных.

Во фреймовых моделях различают процедуры двух типов: процедуры-демоны и процедуры-слуги. *Процедуры-демоны* автоматически запускаются при обращении к слоту, когда происходит определенное событие или возникает определенное условие. Чаще всего таким событием оказывается присваивание слоту значения или его удаление. *Процедуры-слуги* запускаются только при обращении к ним. Типичными примерами процедур-слуг могут служить встроенные в слоты процедуры для определения текущего времени или даты и используемые тогда, когда пользователь их не указывает.

Используемые в настоящее время навигационные ГИС довольно просты по своим функциям, которые, в сущности, сводятся к решению двух задач: определению положения транспортного средства (обычно – автомобиля) с

помощью GPS и картографическому отображению прилегающей территории на экране монитора. Все задачи, возникающие в связи с выбором пути из одной точки земной поверхности в другую точку (за одним исключением), решаются водителем. Единственное исключение состоит в том, что навигационная ГИС может выделить на экране конкретную дорогу, которая, с ее точки зрения, отвечает некоторому критерию оптимальности. В качестве такого критерия обычно выбирается длина дороги.

Однако такой подход к решению задачи выбора пути геоинформационной системой отличается сильным упрощением реальности, и ее (системы) рекомендации довольно примитивны и не всегда могут быть приняты. Например, если кратчайшая между двумя точками дорога является полевой, то это не значит, что в конкретный момент времени по ней можно передвигаться на автомобиле. Очевидно, что такая возможность зависит от нескольких факторов, в том числе от характеристик автомобиля, качества грунтов и погодных условий. Если автомобиль предназначен для движения по хорошим дорогам, грунты глинистые и накануне прошли обильные дожди, то рекомендации ГИС не могут быть приняты и необходимо искать иной путь. Другой пример: если на рекомендуемой дороге находятся мосты, то необходимо сопоставить вес автомобиля и грузоподъемность каждого моста по пути следования. Решение – выбранный путь – может быть разным для легковых автомобилей и тяжелой транспортной техники.

Поэтому *осмысление* ситуации и выбор пути осуществляется человеком. Чтобы навигационная (или транспортная) ГИС могла принимать более обоснованные решения и давать более ценные советы, она должна быть, во-первых, способной осуществлять вывод, во-вторых, обладать солидным запасом знаний о передвижении с помощью транспортных средств, и, в-третьих, ее база данных должна содержать необходимый объем фактических данных о местности и средствах передвижения. Такая ГИС могла бы создаваться на основе фреймовой модели представления знаний.

При ее создании в первую очередь нам потребовалось бы классифицировать все объекты местности с точки зрения возможности их использования тем или иным видом транспорта и представить знания об их свойствах, благоприятствующих или препятствующих передвижению с помощью транспортных средств. Также потребовалось бы ввести общее понятие пути передвижения и его разновидностей: наземного пути и водного пути. В последнем пришлось бы различать реки, озера и моря. Тогда дороги должны быть отнесены к путям передвижения с помощью наземного транспорта, а реки – к водным путям, для передвижения по которым требуется водный транспорт. Необходимо было бы учесть, что реки служат препятствием для наземного транспорта летом, но в зимнее время могут использоваться для передвижения (зимники). Мосты служат для преодоления препятствий в виде водотоков, но в то же время могут служить помехой для водного транспорта.

Рассмотрим в качестве примера на содержательном уровне фрейм, представляющий мост, и некоторые связанные с ним другие фреймы. Во фрейме «мост» необходимо создать указатель на фрейм, содержащий родовое

понятие «сооружение, прокладывающее путь над препятствием». Но возможно, что после некоторых размышлений и консультаций со специалистами (мостовиками) мы создали бы понятие «мостовое сооружение», которое характеризовали бы как путь над препятствием. Поскольку мост обычно входит в комплекс сооружений, называемый мостовым переходом и содержащий наряду с мостом насыпи подходов к нему, берегоукрепительные и регуляционные сооружения, то необходимо ввести указатели для передачи отношения агрегации между мостом и мостовым переходом.

Также необходимо создать указатели на фреймы, описывающие разновидности мостов по типу преодолеваемого препятствия: *собственно мосты* через водотоки, *путепроводы* – через дороги, *виадуки* – через овраги и ущелья. Во фрейме «*виадуки»* потребовалось бы выделить такую их разновидность, как *эстакады*. Но во фрейме «*эстакады*» пришлось бы тем или иным образом представить знания о том, что не все эстакады являются разновидностью виадуков и в общем случае представляют надземное или надводное сооружение мостового типа для пропуска транспортных средств, пешеходов, прокладки инженерных коммуникаций, обеспечения погрузочно-разгрузочных работ и т. д.

Описывая назначение (то есть род прокладываемого пути) мостов, нам пришлось бы выделить *автодорожные*, *городские*, *железнодорожные*, *пешеходные* и *совмещенные* (для нескольких видов транспорта) мосты, а также *мосты – каналы* для пропуска водных путей – и *акведуки* – для целей водоснабжения, *мосты для пропуска нефте- и газопроводов*. Мы вынуждены были бы классифицировать мосты по *материалу* на *деревянные*, *каменные*, *железобетонные* и *стальные* и ввести такие *характеристики* мостов, как *ширина*, *длина*, *грузоподъемность*, *расстояние между опорами*, *высота* низа моста над водой в *межень* (маловодный период продолжительностью не менее 10 дней). Необходимо было бы характеризовать *конструкции мостов*, а также выделить особенности *понтонных* и *разводных* мостов, как и особенности *висячих* мостов (такой мост при входе в нью-йоркскую бухту Веррацано имеет средний пролёт длиной 1 298 м).

Среди других вариантов путей над и/или под препятствиями следовало бы определить паромы, тоннели (также их синонимы) и их разновидности и т. д.

Поскольку качество принимаемых системой решений существенно зависит от объема представленных знаний, постольку важно включить знания о разных видах грунтов, в том числе о *солончаках*, *такырах*, *песках* и не забыть такие детали, как *зыбучие пески* и прочее. В дополнение к обычным характеристикам дорог, отображаемым на топографических картах, пришлось бы вводить и такие, как максимальная и средняя скорость передвижения по дорогам разных типов, максимальные уклоны, выделять горные дороги и т. п. Если система будет содержать знания о возможности передвижения по *морским пляжам*, то потребуется рассчитывать *время приливов и отливов*, то есть положение Луны.

Чтобы система обладала способностью планирования передвижения на длительный период, в базу знаний необходимо включать *статистические*

данные о *погодных условиях* в данной местности (о *температуре*, *количестве выпадающих осадков*, *уровне воды в реках, времени ледостава, средней толщине льда* и т. п.).

Очевидно, что разные варианты такой системы потребовали бы хранения дополнительных специфических знаний. Так, в системе для гражданского населения потребовалось бы хранить информацию о заправочных и авторемонтных станциях, предприятиях общественного питания, местных достопримечательностях и т. п. Система для планирования и управления передвижением войск должна была бы содержать знания о защитных свойствах местности, погрузочно-разгрузочных платформах и др.

Таким образом, можно сделать вывод о том, что хотя знания о местности с точки зрения возможности передвижения по ней не отличаются особой глубиной, но представляют очень обширную область, представление которой на содержательном уровне и осмысление требуют серьезных усилий и системного подхода. Выбор модели представления знаний и ее реализация представляют собой второй этап решения задачи разработки подобной навигационной или транспортной геоинформационной системы.

Понятие фрейма было предложено для обозначения описаний объектов и явлений, обладающих тем свойством, что при удалении из этого описания какой-либо его части происходит потеря свойств, характеризующих объект описания [13]. Обычно фреймом называют некоторую структуру данных для представления стереотипных ситуаций, когда способ группирования множества конкретных ситуаций в стереотипную не всегда может быть строго определен. Обычно представления о стереотипных ситуациях в предметной области формируются у специалистов как результат их индивидуального практического опыта и имеют неформальный характер. Неформальные знания о предметной области можно рассматривать как совокупность понятий, связываемых с конкретными ситуациями. Каждая конкретная ситуация может быть поставлена в соответствие понятию, обозначающему стереотипную ситуацию.

В отличие от понятий, являющихся неформальными знаниями о типичных ситуациях, фреймы представляют собой формализованные знания. С каждым фреймом при этом связывается некоторое понятие, соответствующее определенной сущности предметной области или ее характеристике. Тогда фрейм можно трактовать как структурную единицу или семантический блок представления знаний. Модель знаний о предметной области строится в виде целостной сети фреймов. Поэтому во фреймовых моделях представления знаний естественным образом можно выделить две компоненты: множество фреймов и механизм управления системой фреймов (создания, связывания, модификации и т. п.).

Важной особенностью фреймов является то, что часть слотов может быть незаполненной, то есть не иметь значений. Их заполнение может осуществляться в момент активизации фрейма при выполнении определенных условий. В итоге модель представления знаний приобретает свойство адаптивности на модульном и сетевом уровнях.

Таким образом, фрейм суть декларативно-процедурная структура, то есть объединение декларативных знаний с процедурами, доступ к которым осуществляется непосредственно из фрейма.

Существуют разные концепции, определения и модели фреймов. Различия существуют не только в форме записи и представления фреймов, но и на содержательном уровне. С самой общей точки зрения фрейм рассматривается как формальная структура представления знаний, имеющая вид:

$$f = \{n, (s_1, v_1[, p_1]), ..., (s_m, v_m[, p_m])\},$$

где n – имя фрейма;

s_i – имя слота;

v_i – значение слота;

p_i – присоединенная процедура, которая может отсутствовать.

Имена фреймов играют роль меток компонентов фреймовой сети и могут использоваться в качестве значений некоторых слотов. Таким способом обеспечиваются ссылки из одних фреймов на другие, формирование фреймовой сети в целом, а также возможность включения фреймов друг в друга («принцип матрешки»).

Каждый фрейм может быть охарактеризован как определенный *тип*. Наиболее существенные различия существуют между фреймами-прототипами и фреймами-примерами. *Фрейм-прототип* представляет собой интенсиональное описание определенного множества фреймов-примеров, то есть знания об абстрактных объектах. *Фрейм-пример* – это экстенсиональные знания, описывающие конкретные индивидные объекты. Иначе можно сказать, что фрейм-пример содержит описание отдельного экземпляра индивидных объектов.

Для определения структуры фреймов может использоваться так называемая нотация Бэкуса – Наура. Тогда структура фрейма-примера или фрейма-экземпляра в общем виде может быть представлена выражением

<фрейм>::=[<имя>] <ссылка на прототип> <слот> {<слот>},

где

<слот>::=<имя слота> [<значение слота>][{<процедура>}].

Здесь символ ::= соответствует выражению «по определению есть», квадратные скобки означают, что заключенный в них элемент может отсутствовать, фигурные скобки указывают на некоторое множество, возможно, пустое.

Примером фрейма-прототипа может служить фрейм <город>:

<*город*> (<*название*> <*строка символов* 32>) (<*статус*> <*перечень* {*столица государства, центр субъекта РФ, районный центр, прочие*}> <*функция1*>) (<*год образования*> <*целое* {1, …, 2005}> <*функция2*>) (<*число жителей*> <*целое* {10 000, …, 7 000 000}> <*функция3*>).

В данном примере имя прототипа – *город*. Значением слота *название* может быть любая последовательность символов длиной не более 32. Значением слота *статус* может быть только одно из перечисленных в данном слоте значений. С ним связана процедура *функция1*, которая представляет перечень допустимых

значений. Значением слота *год образования* может быть целое число в диапазоне от 1 до 2005, процедура *функция2* осуществляет проверку этого условия и вызывается автоматически. Значением слота *число жителей* может быть любое целое в диапазоне от 10 000 до 7 000 000, при вводе числа жителей пользователем процедура *функция3* вызывается автоматически и проверяет допустимость введенного значения.

Конкретный фрейм-пример фрейма-прототипа *город* может представлять собой следующую структуру данных:

<*ISA город*> (<*название*> <Новосибирск>) (<*статус*> < *центр субъекта РФ*>) (<*год образования*> <1893>) (<*число жителей*> < 1 500 000>).

Здесь *ISA* («есть») указывает, что данный фрейм является примером (экземпляром) конкретного города.

Для наглядного представления фреймов могут использоваться ориентированные графы с помеченными вершинами и дугами. В таких случаях одна из вершин выделяется для функционального или предикатного символа, а остальные вершины используются для представления их аргументов. Для каждой вершины-аргумента указывается множество допустимых значений (домен), что позволяет рассматривать ее как один из слотов фрейма. Подобная трактовка фреймов близка к понятию факта в семантической сети. На этом основании некоторые авторы считают фреймы одной из разновидностей семантической сети.

Перечень типов фреймов существенным образом зависит от назначения фреймовой системы. Однако можно указать типы фреймов, имеющих более или менее универсальный характер. В действующих системах часто используются такие типы фреймов, как фрейм-соединение, фрейм-назначение, фрейм-закон.

Фреймы-соединения используются для моделирования различных видов связи между объектами предметной области, прототип такого фрейма представлен на рис. 6.23. Данный рисунок следует понимать так, что субъект *S* соединяет объект *O1* с объектом *O2* способом *M*. Дуги обозначены метками падежных отношений: *s* – субъект, *o* – объект, *m* – отношение «каким способом». Вершины (прямоугольники), содержащие в названии префикс *D*, обозначают соответствующие области допустимых значений. В системах геомоделирования такие фреймы могут использоваться, в частности, для описания взаимного положения между различными объектами-компонентами, принадлежащими одному объекту-агрегату.

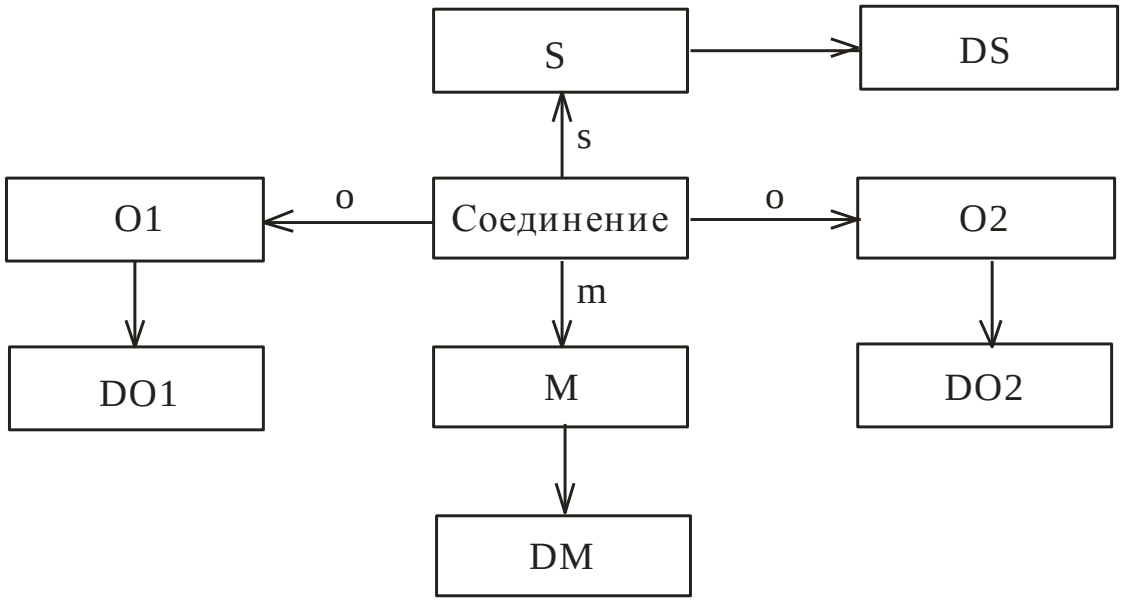


Рис. 6.23. Фрейм-соединение

Фрейм-назначение применяется при описании процессов для указания роли участвующих в них объектов. Пример фрейма-прототипа на рис. 6.24 означает, что судоподъемник SP перемещает судно S с уровня L1 на уровень L2. Как и на рис. 6.23, вершины с префиксом D соответствуют доменам. Дуга i указывает, откуда перемещается судно, а дуга k – куда оно перемещается.

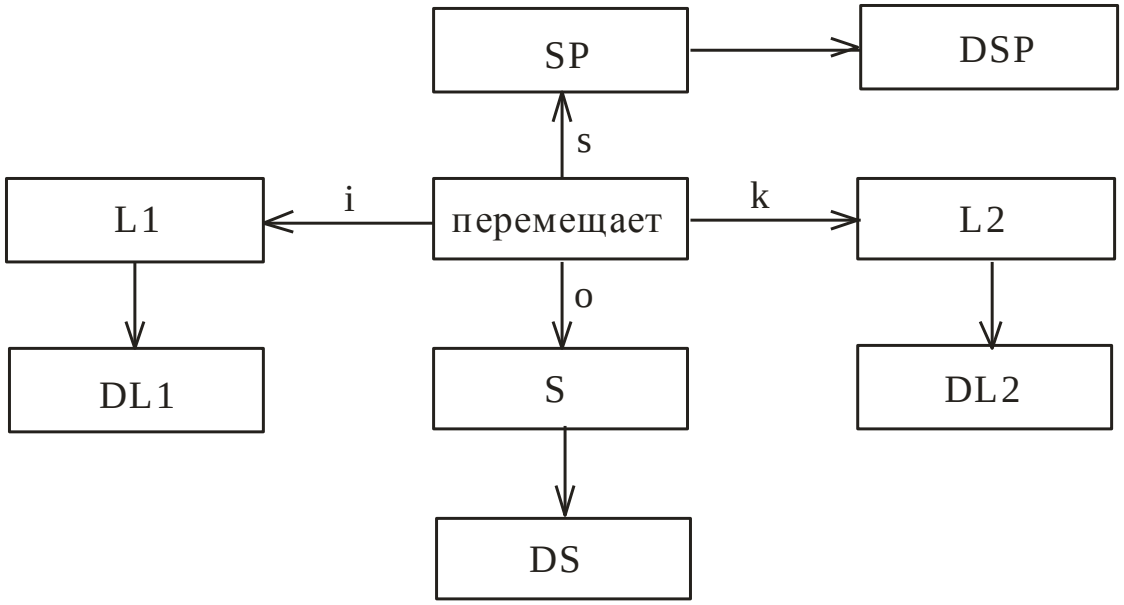


Рис. 6.24. Фрейм-назначение

Фреймы-законы используются для представления аналитических зависимостей, в том числе для представления зависимостей параметров от времени. Фрейм-закон, описывающий ситуацию «Вычислить значение параметра P в момент времени T как функцию F с аргументами $A_1, ..., A_n$» представлен на рис. 6.25. Метки дуг обозначают отношения: f – вид функции; t – время; *arg* – аргумент функции; r – результат.

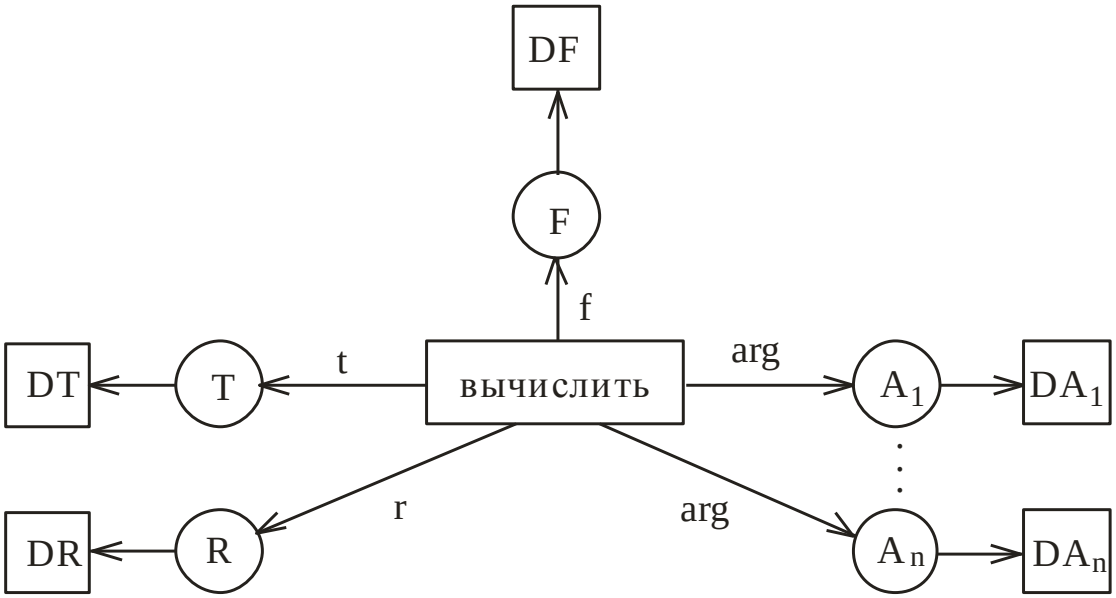


Рис. 6.25. Фрейм-закон

Достоинством систем, использующих фреймы, является то, что вся информация о моделируемом объекте группируется, вследствие чего может извлекаться и обрабатываться как единое целое. Таким образом достигается высокий уровень модульности и, как следствие, относительная легкость создания и модификации знаний, поскольку каждое понятие может описываться независимо от остальных.

6.15. Смешанные представления

Существование трех эвристических моделей знаний (продукционной, семантической и фреймовой) напоминает ситуацию в притче о трех слепых, изучавших слона, когда один из них обследовал хвост слона, другой – ногу, а третий – хобот, и затем споривших о том, что же собой представляет слон. Единая теория экспертных систем отсутствует, и создается впечатление, что каждая из моделей представляет лишь один из аспектов знаний. Наиболее вероятной причиной отсутствия такой теории служит недостаточная изученность мышления.

Наличие некоторых общих свойств, присущих семантическим и фреймовым моделям, дает основания некоторым авторам объединять их под общим названием *сетевых моделей*.

В настоящее время перечисленные типы эвристических моделей «в чистом виде» практически не разрабатываются, а используются их некоторые комбинации, и в каждую модель добавляются некоторые элементы, характерные для моделей других типов.

Представление знаний только в виде семантических сетей часто оказывается неудобным, поэтому в семантические сети включают присоединенные процедуры. С другой стороны, семантические сети расширяют возможности продукционных моделей – основного способа представления знаний в экспертных системах.

6.16. Представление и использование нечетких знаний

Человеческому мышлению в целом присуща нечеткость, тем не менее, люди сравнительно успешно решают задачи в условиях неполноты знаний и той или иной степени неопределенности. Подобные знания принято называть *нечеткими*. Считается, что интеллектуальные системы, близкие по своим возможностям к человеку и обладающие такими характеристиками, как гибкость, широкий кругозор и адаптируемость, должны быть наделены способностью манипулировать нечеткими знаниями. Термин «нечеткие знания» сам является довольно нечетким и общим. Уточняя понятие нечеткости, выделяют такие ее разновидности как недетерминированность выводов, многозначность, ненадежность, неполноту и нечеткость или неточность.

Недетерминированное управление является характерной особенностью систем, основанных на знаниях. Необходимость такого управления вызвана фрагментарностью накопления знаний и невозможностью априорного определения цепи логического вывода при использовании нечетких знаний. Это означает, что решение задач сводится к выбору некоторой цепочки методом проб и ошибок, а в случае неудачи требуется перебор с возвратом для поиска других цепочек.

В системах искусственного интеллекта недетерминированное управление базируется на концепции «*поиска в пространстве состояний*». Данный термин появился в связи с попытками автоматизации игр. Во многих играх имеется конечное число возможных позиций или «состояний», которые могут быть достигнуты из некоторой начальной позиции (например, «крестики и нолики», «пятнадцать»…). Поэтому существует хотя бы теоретическая возможность описания игры путем перечисления всех позиций, которые могут быть достигнуты из начального состояния. Это множество может быть представлено древовидным графом, корневая вершина которого соответствует начальному состоянию. На рис. 6.26 приводятся позиции, которые могут возникнуть в начале игры в «крестики-нолики». Следует учесть, что игровое поле симметрично и одна позиция может быть получена из другой его поворотом.

В данной игре пространство состояний представляет собой множество конкретных ситуаций (положения фишек), сложившихся после нескольких ходов. Конечным является такое состояние, когда игроком достигнута цель (три фишки на одной линии) или когда не осталось незаполненных клеток. Выигрывающие последовательности ходов выявляются в результате прокладки пути от начального состояния до целевого.

Данная игра характерна тем, что в ней используется только один оператор – установка фишки в свободную клетку. В других играх таких операторов может быть несколько, например, в шахматах разным операторам соответствует ход разными фигурами. Поэтому в более сложных играх возникает проблема выбора оператора (или правила) в конкретной ситуации. Общую стратегию выбора таких операторов называют *управляющей структурой*.

Для представления пространства состояний могут использоваться разные способы: строки, векторы, матрицы и графы. При выборе способа

представления пространства состояний следует стремиться к тому, чтобы применение оператора, преобразующего одно состояние в другое, было достаточно простым. Принципиально операторы могут быть заданы с помощью таблицы, в которой каждому «входному» состоянию соответствует некоторое число «выходных» состояний. Однако, для задач с большим числом состояний такой способ не пригоден.

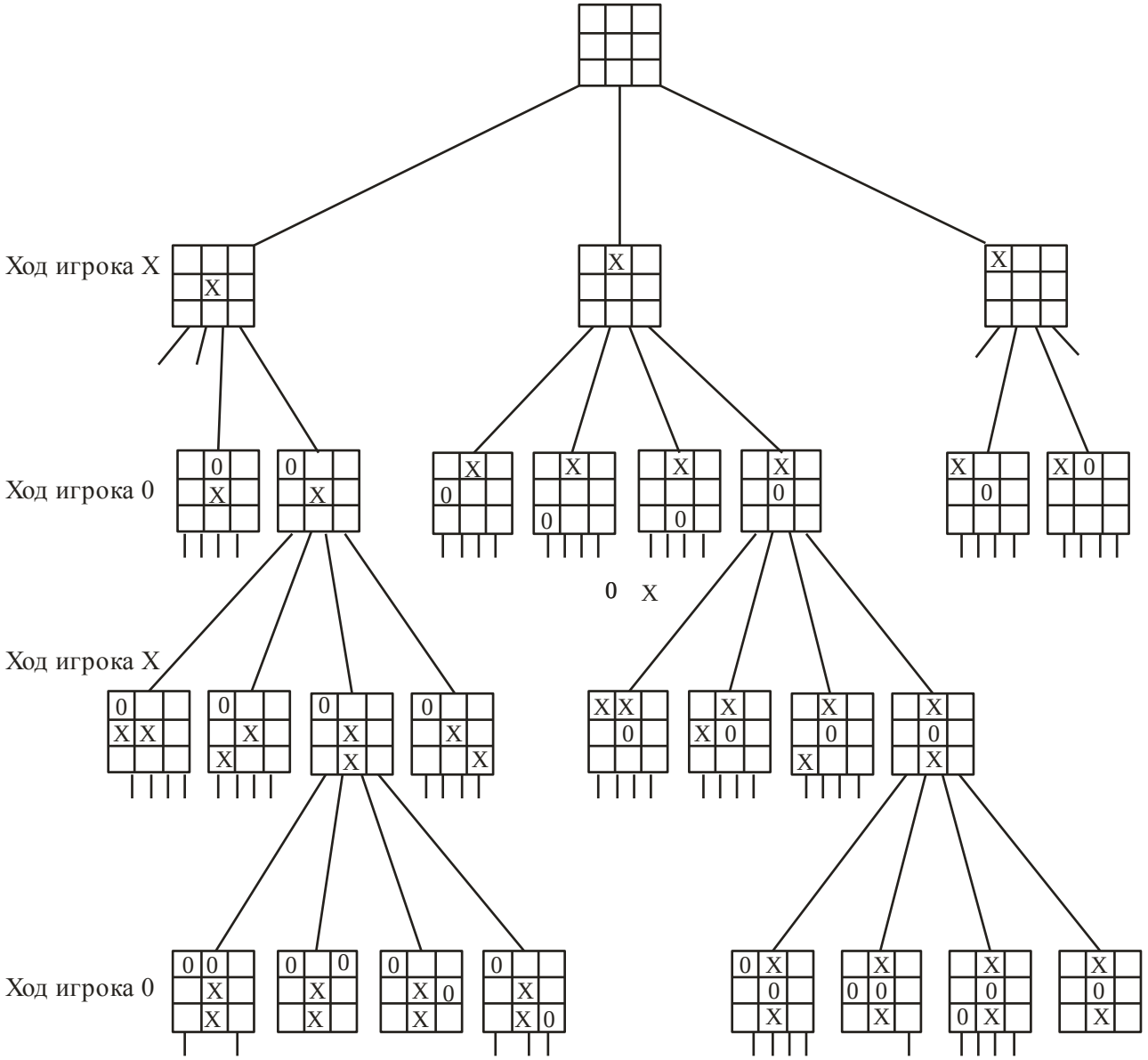


Рис. 6.26. Начало игры «крестики-нолики»

Поиск решения в пространстве состояний сводится к нахождению последовательности операторов, преобразующих входные состояния в выходные таким способом, что после конечного числа таких операторов достигается некоторое целевое состояние. Тогда формально представление задачи в пространстве состояний можно определить как тройку (S, F, G), где S – множество начальных состояний, F – множество операторов, отображающих одно состояние в другое (то есть отображение пространства состояний в самого себя), и G – множество целевых состояний.

При изображении пространства состояний в графическом виде обычно начальное состояние помещают внизу, а конечные состояния – вверху, так как внимание концентрируется на целях. Исходных и конечных состояний может быть несколько. *Прямым направлением* поиска решения считается движение от начального состояния к цели, называемое также направлением «снизу вверх». *Обратным направлением* называют движение от цели к начальному состоянию (или «сверху вниз»).

Вне зависимости от ориентации пространства состояний его просмотр может осуществляться «сначала вглубь» или «сначала вширь». При поиске «сначала вширь» производится порождение всех возможных вариантов на заданном уровне, затем – всех альтернатив на следующем уровне и т. д. При поиске «сначала вглубь» дерево просматривается из заданного состояния на всю глубину до исчерпания последовательности «преемственных» состояний на выбранном пути. После этого просматриваются альтернативные пути, ведущие вглубь дерева.

Простейшая стратегия при выборе пути к цели состоит в переборе всех вариантов. Но даже в такой простой игре, как «крестики-нолики», существует 9! или 362 880 вариантов игры. За счет симметрии их количество можно сократить примерно до 60 000. В более сложных играх количество вариантов резко возрастает, то есть происходит уже упоминавшийся комбинаторный взрыв. В шахматах после первых пяти ходов число возможных вариантов составляет 10^{15}, а после 40 ходов (средняя продолжительность партии) – 10^{120} комбинаций. Для сравнения указывалось, что с момента зарождения вселенной прошло менее 10^{80} секунд [22].

Очевидно, что по соображениям эффективности требуется тем или иным способом определять пути, по которым поиск решения должен осуществляться в первую очередь. В играх и многих ситуациях при обработке знаний часто не представляется возможным со всей определенностью установить последовательность действий (операторов) или применяемых правил, ведущую к успеху. Но нередко удается установить правила, увеличивающие *вероятность* достижения цели. Такие правила существенным образом зависят от предметной области и называются *эвристическими*.

Для применения того или иного правила требуется оценка его эффективности в виде значения некоторой «оценочной функции». Однако определение таких функций при решении многих реальных задач уже само по себе является сложной проблемой. В результате многочисленных попыток найти решение данной проблемы были предложены две не противоречащие здравому смыслу универсальные стратегии:

– организация пространства задачи таким образом, чтобы на начальной стадии поиска решения было осуществимо прогнозирование возможности достижения цели при движении по каждому пути и игнорирование малоперспективных направлений;

– построение пространства задачи в виде подпространств с минимальными взаимосвязями или без таковых, что дает возможность их независимого анализа и решения.

Проблема *многозначности* обычно трактуется как проблема *многозначности интерпретации* при решении задачи понимания текстов на естественных языках и распознавании изображений. Проблема многозначности существует при машинном переводе с одного этнического языка на другой. В одной из книг была рассказана история о программе для перевода с русского языка на английский, для контроля осуществлявшей обратный перевод. Когда в машину ввели фразу «Плоть слаба, а дух крепок», то обратный перевод гласил «Мясо мягкое, а водка крепкая». Впрочем, нет гарантий, что эта правдоподобная история не была придумана для демонстрации проблемы разрешения многозначности.

При понимании текстов причинами многозначности являются омонимия естественных языков, многозначность подчиненности слов в предложении, многозначность местоимений и т. д. Американский специалист по вычислительным системам Б. Байцер писал, что ему приходилось участвовать в разработке, в которой термин «канал связи» имел более шести значений [23]. Устранение такой многозначности обычно достигается за счет семантических ограничений – выбором определенного контекста.

Еще одним примером неоднозначности могут служить две фразы: «На футбольном матче присутствовало много зрителей» и «Во время матча было забито много голов». Во-первых, если вместимость стадиона более 100 000 зрителей, то нельзя сказать, что 50 000 – это много. Но 5 голов за матч – это много, поскольку в среднем за одну игру забивают 2–3 гола. Для разрешения этой неоднозначности можно использовать теорию нечетких множеств. Во-вторых, еще неизвестно, как система проинтерпретировала бы слово «голов». Не исключено, что она могла понять его как форму слова «голова», и решить, что, если головы можно разбивать, то, наверное, их можно и «забивать». Интеллектуальная система еще больше бы укрепилась в своих выводах, если до этого ей приходилось обрабатывать фразу «Во время эпидемии … было забито много голов крупного рогатого скота». А система с фантазией могла бы предположить, что скот забивают на стадионах во время футбольных матчей.

На этом неоднозначности с интерпретацией «забивания голов» не кончаются. Так, интеллектуальная система должна понять, что фраза «забивать головы студентов интеллектуальными системами» не означает их физического уничтожения.

Относительно неоднозначности в данном примере можно сказать, что проблема интерпретации слова «много» имеет количественный аспект, а интерпретация словосочетания «забито голов» – качественный, так как при разных его интерпретациях изменяется смысл высказываний.

При распознавании изображений также часто возникает многозначность интерпретации их элементов. Так, на рис. 6.27 изображение куба можно интерпретировать двумя способами: ребро *AB* вертикально (вершина *A* внизу) и ребро *AB* горизонтально (вершина *A* ближе наблюдателю). Избавление от многозначности изображений достигается путем использования более широких пространственных отношений и других довольно специфических методов, одним из которых является метод релаксации.

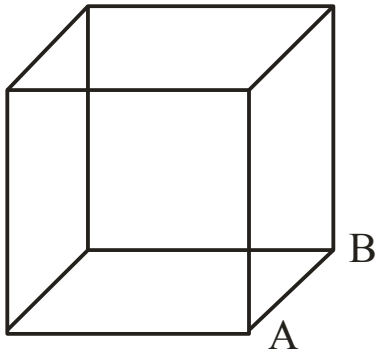


Рис. 6.27. К задаче интерпретации изображений

В экспертных системах часто возникает необходимость делать выводы на основе *недостоверных фактов и знаний*. Использование теории вероятностей, основанной на законах Байеса, признается не вполне логичным. Поэтому для оценки выводов на основе ненадежных фактов и знаний были предложены методы, не имеющие строгого теоретического обоснования, но результативные. Разработка более поздних систем проходила под влиянием этих методов.

Сложная задача может быть разбита на подзадачи, каждая из которых также может быть разбита на более простые подзадачи и т. д. Поэтому вся задача в целом может быть представлена в виде дерева. Знания и факты, полученные при решении подзадач и используемые в качестве исходных данных при решении задач верхних уровней, порождаются фрагментарно. Если полученные знания характеризуются только двумя истинностными значениями 0 и 1, то связь между подзадачами может быть выражена с помощью дизъюнкции и конъюнкции. Достоверность нечетких данных характеризуется промежуточными значениями между истиной и ложью. В задачах с ненадежными исходными данными кроме связок И и ИЛИ используется играющая важную роль комбинированная связь, обозначаемая как КОМБ. Такие связи независимо подкрепляют или опровергают цель на основании нескольких доказательств. Дерево, содержащее связи И, ИЛИ и КОМБ, называют деревом И/ИЛИ/КОМБ (рис. 6.28).

Вершины называют соответственно И-вершинами, ИЛИ-вершинами и КОМБ-вершинами. Среди вершин выделяют следующие типы:

– *заключительные* – вершины, соответствующие элементарным задачам, то есть таким, решение которых очевидно;

– *тупиковые* – вершины, не имеющие дочерних вершин и не являющиеся заключительными;

– *разрешимые* – вершины, которым соответствуют разрешимые задачи;

– *неразрешимые* – вершины, соответствующие неразрешимым задачам.

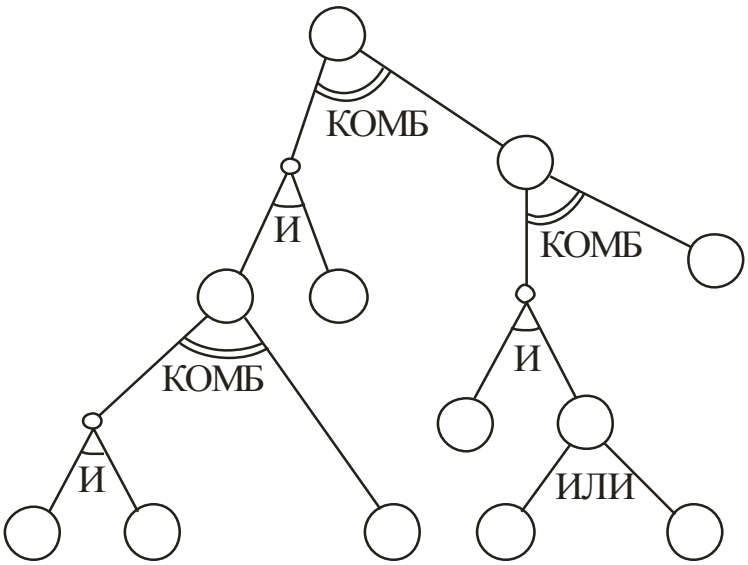


Рис. 6.28. Пример дерева И/ИЛИ/КОМБ

Иногда используется несколько иной способ наглядного представления задач (рис. 6.29). В вершинах И/ИЛИ задачи разбиваются на группы, внутри которых они связаны отношением И, а между группами существует отношение ИЛИ. При существовании отношения конъюнкции между подзадачами используется дуга, соединяющая эти вершины (вершины *G* и *H, D* и *E, J* и *K* на рис. 6.29). Тогда, чтобы получить решение какой-либо задачи, необходимо и достаточно решить все подзадачи любой ее группы. Но чаще дерево задач представляется в таком виде, когда между подзадачами (вершинами) существуют отношения только одного вида (или И, или ИЛИ), для чего вводятся дополнительные вершины.

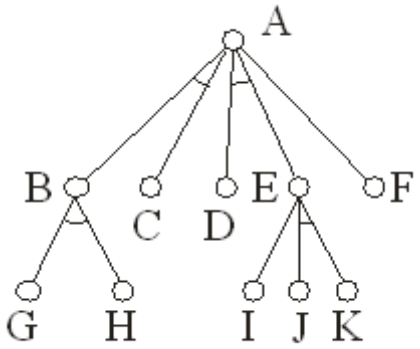


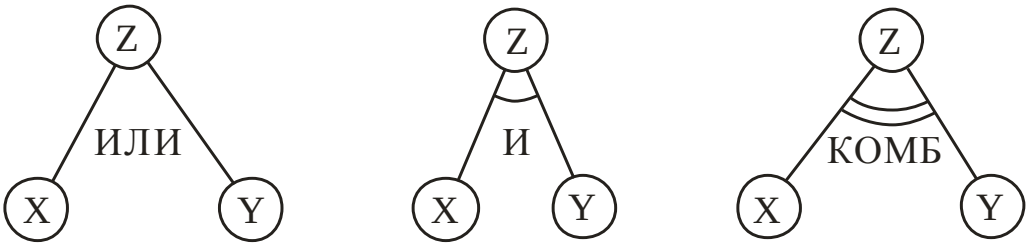
Рис. 6.29. Другое представление задач

Если для представления знаний используются правила, то можно ввести правила для характеристики надежности выводов, получаемых с использованием связок И, ИЛИ и КОМБ (рис. 6.30).

Допустим, что известны степени надежности утверждений X и Y, полученных как результат вывода или наблюдений. Требуется определить степень надежности утверждения Z как функцию от степени надежности X и Y.

Кроме того, само правило, на основании которого делается вывод Z, также может быть ненадежным. Степени надежности данных и правил характеризуются истинностными значениями от 0 (ложь) до 1 (истина). Тогда степень доверия к Z C_Z можно представить как

$$C_Z = \min\{C_{предпосылки}, C_{правила}\}.$$



ЕСЛИ X или Y
ТО Z с C1

ЕСЛИ X
ТО Z с C11

ЕСЛИ Y
ТО Z с C12

ЕСЛИ X и Y
ТО Z с C2

ЕСЛИ X
ТО Z с C31

ЕСЛИ Y
ТО Z с C32

Рис. 6.30. Степени надежности доказательств

Если имеется несколько посылок, связанных с помощью ИЛИ, то степень надежности заключения принимают равной максимальному значению надежности из всех значений, характеризующих отдельные посылки

$$C_{предпосылки} = \max\{C_{предпосылки\_1}, ..., C_{предпосылки\_n}\}.$$

Если *X* и *Y* не могут выполняться одновременно, то степень доверия к *Z* считается равной степени доверия к соответствующей посылке. Если посылки связаны с помощью И, то степень доверия заключению считают равной степени доверия к посылке, значение которой минимально среди отдельных посылок, то есть,

$$C_{предпосылки} = \min\{C_{предпосылки\_1}, ..., C_{предпосылки\_n}\}.$$

Для определения степени доверия заключению, полученному с помощью КОМБ, в различных системах используются разные методы.

Среди других проблем, связанных с представлением и использованием нечетких знаний, в качестве самостоятельных классов выделяют проблемы, обусловленные *немонотонностью выводов* и существованием нечетких множеств. Во-первых, немонотонность выводов вызвана тем, что наряду с правилами существуют и исключения из них. Во-вторых, как уже отмечалось выше, нельзя все предвидеть и представить соответствующим образом. Таким же образом, можно перечислить верные знания в конкретной проблемной области, но перечислить (бесконечное) множество неверных знаний в принципе невозможно. Поэтому в базах знаний принято представлять только верные знания, а все, что не определено, считается априори ложным. Данное

соглашение называют «*гипотезой закрытого мира*» и применяют для неопределенных знаний. В классической логике исходят из предположения о том, что набор аксиом полон, и правильный вывод не изменяется при добавлении новой аксиомы, то есть имеет место свойство, называемое монотонностью. Некоторые авторы автоматизацию немонотонных рассуждений, основанных на здравом смысле, называют одной из самых больших проблем в искусственном интеллекте.

6.17. Инструментальные средства

С помощью универсальных языков программирования может быть проинтерпретирована любая модель представления знаний, так как в этих языках имеется декларативная часть – описание данных – и процедурная часть – последовательность операций над данными [13]. Более богатые возможности в этом отношении имеют универсальные объектно-ориентированные языки программирования, отличительной особенностью которых является наличие средств для создания классов – абстрактных типов данных. Однако использование универсальных языков программирования для обработки знаний характеризуется высокой сложностью и большой трудоемкостью.

Для того чтобы повысить эффективность программной реализации моделей представления знаний, было разработано большое число специальных языков представления знаний (ЯПЗ). К особенностям таких языков относят:

- более мощные, чем в обычных языках программирования, средства описания типов данных и процедур управления;
- встроенные механизмы представления, поиска и обработки информации;
- средства построения дедуктивных алгоритмов.

Всю совокупность языков представления знаний подразделяют на три группы.

1. *Языки обработки символьной информации* (LISP, SNOBOL и др.). Такие языки считаются наиболее применимыми для некоторых задач искусственного интеллекта в связи с тем, что в основе интеллектуальной деятельности лежит способность манипулировать символами.

2. *Языки, ориентированные на поиск решений в пространстве состояний и доказательство теорем* (PLANNER, QLISP …), имеют достаточно простые методы определения данных, но сложные методы обработки данных. Поэтому в центре внимания языков этой группы находится проблема эффективности управления выполнением программы. Представитель языков этой группы PLANNER удачно сочетает возможности языка LISP с развитым механизмом сопоставления образцов, поиска с возвратом, вызова процедур по образцу. *Сопоставление образцов* позволяет создавать более лаконичные и более простые программы, чем на языке LISP. *Механизм поиска с возвратом* – это такой способ выполнения программы, когда делаются различные пробные шаги, но если некоторые из них приводят к тупиковой ситуации, то программа от них отказывается. Механизм поиска с возвратом оказывается полезным при решении задач перебора, являющихся типичными при поиске в пространстве

состояний. *Вызов процедур по образцам* используется для реализации процедур поиска методом сведения задач к подзадачам.

Среди языков этой группы выделяется продекларированный как язык логического программирования PROLOG, в том числе – полярным отношением к нему. В литературе можно встретить как самые положительные, так и негативные отзывы о нем, вплоть до рекомендаций всячески избегать его использования и даже забыть о нем [21]. Критики PROLOGa высказывают следующие претензии в его адрес:

- слабые средства защиты от ошибок в написании программ;
- пользователю необходимо понимать *детали реализации* встроенного механизма возвращения;
- порядок операторов имеет существенное значение, что противоречит концепции логического программирования;
- многие встроенные предикаты дают побочные эффекты, что делает невозможным его применение при параллельном программировании;
- PROLOG предоставляет реляционную базу данных, но эта база существует только в оперативной памяти, в результате чего возрастают требования к ее объему;
- в PROLOG изначально встроен поиск в глубину, поэтому реализация поиска в ширину требует изощренности.

В завершение этой критики Р. Форсайт называет его «мечтой лесоруба» и говорит, что «лишь преданная этому языку элита в состоянии овладеть его элегантными суперсложностями» [21, с. 22].

3. *Языки представления знаний* (FRL, KRL…), называемые также языками второго поколения и фреймовыми языками, характеризуются следующими особенностями:

- данные представляются как многоуровневая фреймовая структура;
- абстрактная модель предметной области представляет собой иерархическую структуру множества понятий;
- конкретная модель ситуации представляется совокупностью взаимосвязанных экземпляров понятий (индивидных объектов);
- сопоставление образцов осуществляется с помощью семантического сопоставления, являющегося развитием синтаксического.

Языки, называемые фреймовыми, имеют некоторые общие черты с объектно-ориентированными языками программирования, но отличаются высокой универсальностью, поскольку дают возможность как представления знаний в виде сетевой структуры, так и эффективной разработки программ управления выводом с применением присоединенных процедур. Платой за подобного рода универсальность может служить дополнительная нагрузка на пользователя. Вообще же фреймовые языки относят к языкам, ориентированным на специалистов по обработке знаний, а не на конечных пользователей. Считается, что фреймовые языки являются средством для разработки сложных приложений.

Главной особенностью применения языков фреймового типа является простота разработки программ для решения интеллектуальных задач. Однако людьми, не знакомыми с проблемами обработки знаний, они воспринимаются как разновидность ставших обычными объектно-ориентированных языков программирования.

Как показала практика, при разработке экспертных систем методология традиционного программирования оказывается неэффективной [11]. Причина неудач заключается в неформализованном характере решаемых задач, отсутствии завершенной теории экспертных систем и методологии их проектирования. Проблемы разработки ЭС начинаются уже с выбора инструментальных средств.

Для разработки систем, основанных на знаниях, в общем случае могут использоваться следующие инструментальные средства:

- процедурные языки программирования;
- языки, ориентированные на разработку экспертных систем или языки инженерии знаний;
- средства автоматизации разработки и модификации экспертных систем;
- пустые ЭС или ЭС-оболочки, не содержащие знаний о какой-либо предметной области.

В данном перечислении инструментальные средства упорядочены по убыванию трудоемкости разработки. *Универсальные языки программирования*, широко используемые при автоматизации решения алгоритмических задач, не содержат эффективных адекватных средств для реализации вывода на основе знаний и поэтому характеризуются как языки достаточно низкого уровня.

Языки, ориентированные на обработку знаний, дают возможность существенно повысить производительность труда разработчиков, но системы, реализованные с их применением, характеризуются меньшей эффективностью, чем системы, созданные на универсальных языках программирования.

Инструментальные *средства автоматизации разработки ЭС* содержат готовые компоненты экспертных систем. Поэтому разработчик ЭС имеет возможность конструировать систему из представленного набора компонентов и отказаться частично или полностью от программирования компонентов ЭС. Среди инструментальных средств этого типа выделяют средства, автоматизирующие процесс построения ЭС, и средства, автоматизирующие процесс приобретения знаний.

Использование пустых ЭС может оказаться самым коротким путем к созданию нужной системы обработки знаний. Однако, на этом пути возможно возникновение серьезных проблем. Во-первых, управляющая стратегия, используемая выбранной ЭС-оболочкой, может быть неадекватной методам рассуждений экспертов в автоматизируемой проблемной области, что может приводить к потере эффективности и, что еще хуже, к получению неправильных выводов. Во-вторых, язык представления знаний в используемой базовой ЭС может оказаться непригодным для конкретного приложения.

Поскольку разработка экспертных систем признана сложной задачей, постольку получила признание концепция быстрого прототипирования ЭС,

иногда применяемая и при создании обычного программного обеспечения. Ее суть состоит в том, что первоначально не ставится задача разработки окончательного варианта такой системы. Первоначальная цель состоит в максимально быстром получении работоспособной версии системы, которая может быть неэффективной и функционально неполной, но способной решать наиболее типичные задачи проблемной области. Создание окончательного варианта ЭС рассматривается как итерационный процесс, состоящий из нескольких этапов. Окончание каждого этапа означает создание определенного более совершенного варианта системы.

По степени готовности ЭС к функционированию различают следующие версии ЭС или стадии существования: демонстрационный прототип, исследовательская система, действующая система, промышленная система и коммерческая система.

Демонстрационный прототип – это версия экспертной системы, способная решать часть задач проблемной области. Ее назначение состоит в проверке и демонстрации пригодности методов искусственного интеллекта для решения типичных задач в сфере ее применения. Такая версия обладает минимальными средствами общения с пользователем и отличается крайне низкой эффективностью. База знаний демонстрационного прототипа может содержать до одной сотни правил. По оценкам специалистов в области ЭС, при использовании развитых инструментальных средств на разработку демонстрационного прототипа требуется в среднем около 3 месяцев.

Исследовательские системы создаются для проверки принципиальных решений и предназначаются для решения более широкого класса задач, характерных для проблемной области, но отличаются неустойчивым функционированием. База знаний исследовательской ЭС содержит обычно несколько сотен правил (200–500). Время разработки исследовательской системы оценивается в 1–2 года.

Действующие системы представляют собой следующий уровень, отличаются более тщательной проработкой основных вопросов и, как правило, решают все задачи автоматизируемой проблемной области. Но их недостатком являются чрезмерные требования к памяти и все еще низкое быстродействие. Объем базы знаний в действующих системах увеличивается до 500–1 000 правил, а на их реализацию, по оценкам, требуется до 2–3 лет.

Промышленная ЭС – это версия экспертной системы, решающая все необходимые задачи при минимуме затрат времени и памяти ЭВМ. Разработка такого варианта заключается преимущественно в переписывании текстов программ с использованием более эффективных инструментальных средств, обычно – универсальных языков программирования. База знаний обычно включает до 1 000–1 500 правил. Реализация промышленной ЭС может потребовать от 2 до 4 лет.

Коммерческая система представляет собой продукт, готовый к использованию широким кругом пользователей, то есть предмет продаж. База знаний на этой фазе разработок может составлять до 3 000 правил. Разработка же коммерческой системы требует порядка 3–6 лет. Известны прецеденты, когда

экспертные системы разрабатывались до 16 лет (DENDRAL), что позволило достичь такого уровня, когда качество предлагаемых системой решений в среднем оказывалось выше, чем у специалистов.

6.18. Этапы разработки ЭС

Считается, что при создании экспертных систем необходимо соблюдение следующих общих принципов:

- базы знаний должны быть отделены от механизма логического вывода;
- представление знаний должно быть по возможности единообразным, при этом наиболее предпочтительной формой являются правила продукций;
- необходимо стремиться к созданию максимально простой управляющей структуры;
- система должна обладать функцией разъяснения своих рассуждений в виде, понятном пользователю;
- наиболее благоприятной областью применения являются задачи, решение которых требует использования больших объемов эмпирических ассоциативных знаний.

Представление знаний на основе правил позволяет расширять знания по мере развития системы и не накладывать серьезных ограничений на их содержание. Появляется возможность объединения знаний, требующих глубокого теоретического понимания, с простыми знаниями, полученными чисто эмпирическим путем. Использование простой управляющей структуры позволяет упростить реализацию «объяснительной» компоненты. Системы, объясняющие ход своих рассуждений, вызывают большее доверие со стороны пользователей.

В результате анализа различных экспертных систем были сформулированы перечисляемые ниже критерии выбора предметных областей, в которых применение систем, основанных на знаниях, оказывается удачным.

1. Предметная область должна быть сравнительно узкой и характеризоваться небольшим объемом знаний, основанных на здравом смысле.
2. Решаемые задачи не должны быть ни слишком легкими, ни слишком трудными для специалистов; количество используемых понятий не должно превышать несколько сотен.
3. Решаемые задачи должны допускать их четкую формулировку.
4. Для решения задач необходимо привлечение знаний экспертов, способных объяснять ход своих рассуждений.

Кроме того, иногда называют еще два критерия. Так, весьма желательным является наличие в выбранной предметной области некоторого формализованного аппарата. При его отсутствии реализация экспертной системы возможна, но требует больших затрат. Второй критерий состоит в выборе такой предметной области, описание которой может осуществляться с помощью вербальных средств.

Разработка каждой экспертной системы имеет свою специфику, поэтому можно говорить о более или менее общих этапах, представленных на рис. 6.31:

идентификация, концептуализация, формализация, выполнение и опытная эксплуатация.

Этап идентификации. На данном этапе решаются задачи:

- идентификации проблемы;
- определения целей разработки;
- определения потребности в ресурсах и состава разработчиков.

Идентификация проблемы заключается в подготовке ее вербального описания. Такое описание должно содержать общие характеристики проблемы, выделенные подпроблемы, основные понятия и отношения, входные данные, предполагаемые решения, знания, релевантные решаемой проблеме. Прежде всего необходимо осуществить оценку пригодности методов искусственного интеллекта к решению задач в автоматизируемой проблемной области. Предпосылками возможности применения таких методов служат: качественный характер решений, относительно узкий класс и умеренная сложность решаемых задач. Задачи, решаемые экспертом, не должны быть ни слишком легкими, ни слишком трудными. Общее число используемых понятий может содержать несколько сотен.

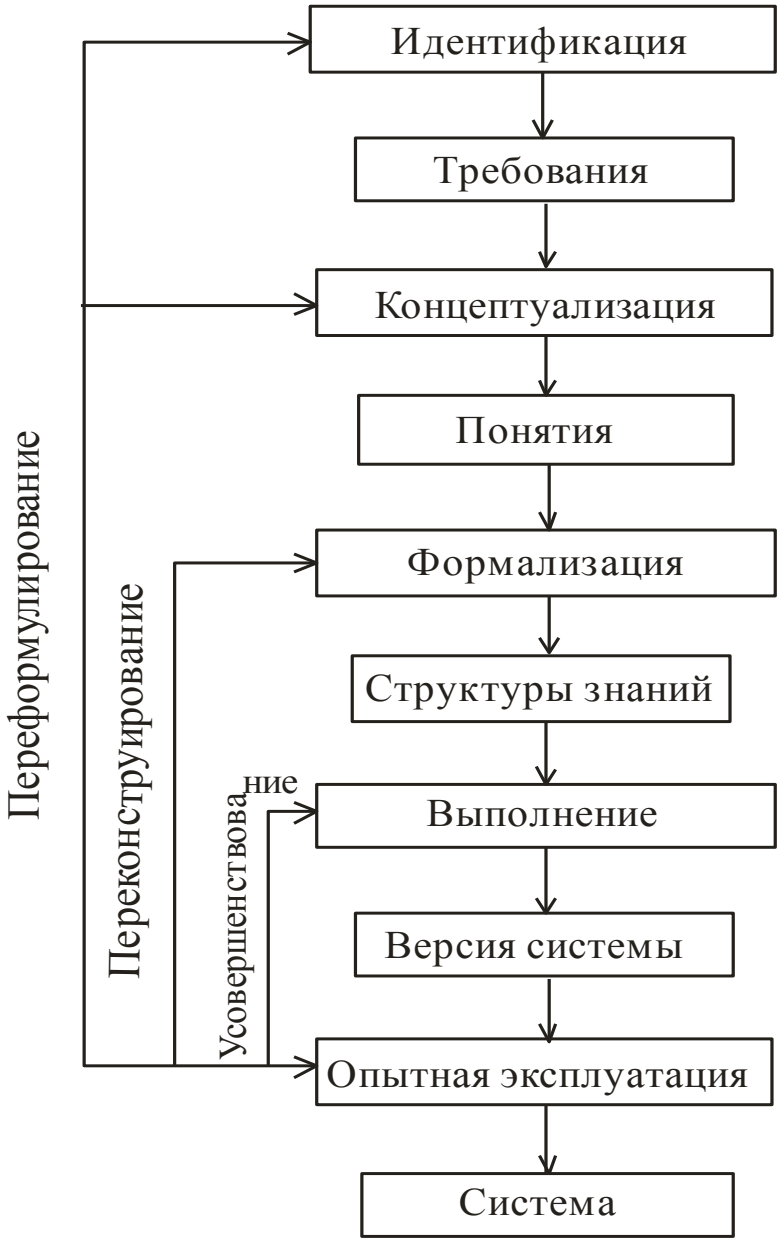


Рис. 6.31. Этапы разработки ЭС

Определение целей разработки заключается в их формулировании в явном виде. При этом цели разработки должны различаться от решаемых задач. Целями разработки могут быть улучшение качества решений в области экспертизы, автоматизация рутинных процессов, сокращение сроков решения задач и т. п.

В литературе настойчиво рекомендуется включать в коллектив разработчиков специалистов трех категорий:

– специалистов в автоматизируемой проблемной области, называемых *экспертами*;

– специалистов по разработке экспертных систем, называемых *инженерами по знаниям*;

– и *программистов*, задачей которых является согласование и разработка программных средств. Особенно подчеркивается обязательное привлечение к подобным разработкам инженеров по знаниям.

Этап концептуализации. Целью данного этапа является уточнение описания проблемы, полученного на предыдущем этапе. При этом определяются следующие особенности проблемы: используемые типы данных, входные и выходные (выводимые) данные, используемые стратегии и гипотезы, виды отношений между объектами (или понятиями) предметной области, ограничения на решения задач, состав используемых знаний. На данном этапе рекомендуется построить содержательную модель рассуждений эксперта при решении хотя бы одной задачи и не осуществлять оценку корректности и полноты разрабатываемой системы, а как можно быстрее переходить к следующему этапу.

Этап формализации. В процессе формализации основные понятия и отношения, полученные на этапе концептуализации, выражаются на некотором формальном языке. Результатом данного этапа должно быть описание решения поставленной проблемы на формальном языке. Иначе, на этапе формализации решается задача определения состава и способа представления декларативных и процедурных знаний. Кроме того, осуществляется оценка пригодности инструментальных средств и необходимость выполнения оригинальных разработок.

Наибольшее влияние на процесс формализации оказывают: особенности структуры пространства поиска решений; модели, используемые для получения решения; особенности данных, описывающих состояние предметной области. В процессе определения структуры пространства поиска осуществляется формализация используемых понятий и отношений. При этом, в частности, выясняется структура понятий, необходимость представления причинно-следственных, пространственно-временных и других отношений в явном виде, необходимость использования коэффициентов неопределенности и т. д.

Описание данных должно включать их следующие свойства: достоверность (надежность, точность) / недостоверность, полнота (или достаточность) / неполнота, согласованность / противоречивость, избыточность / неизбыточность, определенность / неопределенность, а также способ и стоимость их получения.

Этап выполнения. Задачей данного этапа служит получение работающего прототипа ЭС путем программирования и/или использования готовых компонент и наполнения базы знаний. Считается обычной ошибкой перенос работ по наполнению базы знаний (наиболее трудоемких) на более поздние сроки, на период после окончания работ, связанных с программированием. Весьма желательным является раннее наполнение базы знаний, поскольку накопление знаний часто влечет за собой перепрограммирование. Поэтому принято считать, что наполнение баз знаний должно начинаться, когда появляется возможность создания простейших правил и использования примитивных управляющих структур.

Получение первой работающей версии ЭС позволяет приступить к оценке пригодности выбранных методов для решения хотя бы части задач, и, в случае неудачи, как можно раньше начать процесс переосмысления или переформализации предметной области. Реализация программного обеспечения

ЭС носит итеративный характер, а ее трудоемкость зависит от сложности проблемной области, гибкости используемого формализма представления знаний и степени соответствия управляющего механизма классу решаемых задач.

Этап опытной эксплуатации. Целью данного этапа является получение заключения конечных пользователей о принципиальной пригодности системы для решения задач в их проблемной области. В первое время своего существования системы отличаются определенной недоработанностью. Первое впечатление пользователей о любом программном продукте отличается устойчивостью, поэтому ЭС должна передаваться в опытную эксплуатацию достаточно, хотя и не полностью, отлаженной, и должны приниматься определенные превентивные меры по подготовке пользователей к восприятию системы и работы с ней.

В результате опытной эксплуатации пользователи должны вынести свое суждение, во-первых, о полезности системы и, во-вторых, об удобстве ее использования. Под *полезностью* ЭС в данном случае понимается ее способность решать поставленные задачи, а под *удобством* – естественность взаимоотношений пользователя с системой. По результатам опытной эксплуатации выносится «судьбоносное» решение для дальнейшего существования ЭС:

- либо осуществляется переформулирование задачи;
- либо переконструирование системы;
- либо ее совершенствование;
- либо ее подготовка к реализации как готового программного продукта.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. – СПб.: Питер, 2000. – 384 с.
2. Искусственный интеллект. – В 3-х кн. Кн. 1. Системы общения и экспертные системы: Справочник / Под ред. Э.В. Попова. – М.: Радио и связь, 1990. – 464 с.
3. Искусственный интеллект. – В 3-х кн. Кн. 2. Модели и методы: Справочник / Под ред. Д.А. Поспелова. – М.: Радио и связь, 1990. – 304 с.
4. Искусственный интеллект. – В 3-х кн. Кн. 3. Программные и аппаратные средства: Справочник / Под ред. В.Н. Захарова, В.Ф. Хорошевского. – М.: Радио и связь, 1990. – 368 с.
5. Кандрашина Е.Ю., Литвинцева Л.В., Поспелов Д.А. Представление знаний о времени и пространстве в интеллектуальных системах. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 328 с.
6. Кузнецов В.Е. Представление в ЭВМ неформальных процедур. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 160 с.
7. Любарский Ю.Я. Интеллектуальные информационные системы. – М.: Наука. Гл. ред. физ.-мат. лит., 1990. – 232 с.
8. Нейлор К. Как построить свою экспертную систему. – М.: Энергоатомиздат, 1991. – 286 с.
9. Осуги С., Саэки Ю. Приобретение знаний. – М.: Мир, 1990. – 304 с.
10. Перспективы развития вычислительной техники. В 11 кн.: Справ. пособие. Кн. 2. Кузин Е.С. и др. Интеллектуализация ЭВМ. – М.: Высш. шк., 1989. – 159 с.
11. Попов Э.В. Экспертные системы: Решение неформализованных задач в диалоге с ЭВМ. – М.: Наука. Гл. ред. физ.-мат. лит., 1987. – 288 с.
12. Реальность и прогнозы искусственного интеллекта: Сб. статей. – М.: Мир, 1987. – 247 с.
13. Робототехника и гибкие автоматизированные производства. В 9-ти кн. Кн. 6. Назаретов В.М., Ким Д.П. Техническая имитация интеллекта. – М.: Высш. шк., 1986. – 144 с.
14. Рубашкин В.Ш. Представление и анализ смысла в интеллектуальных информационных системах. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 192 с.
15. Управление, информация, интеллект. Под ред. А.И. Берга и др. – М.: Мысль, 1976. – 386 с.
16. Уэно Х. и др. Представление и использование знаний. – М.: Мир, 1989. – 220 с.
17. Философский словарь / Под ред. И.Т. Фролова. – М.: Политиздат, 1981. – 445 с.
18. Хант Э. Искусственный интеллект. – М.: Мир, 1978. – 558 с.
19. Цаленко М.Ш. Моделирование семантики в базах данных. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 288 с.
20. Шалютин С.М. Искусственный интеллект: Гносеологический аспект. – М.: Мысль, 1985. – 189 с.

21. Экспертные системы. Принципы работы и примеры / Под ред. Р. Форсайта. – М.: Радио и связь, 1987. – 224 с.

22. Элти Дж., Кумбс М. Экспертные системы: концепции и примеры. – М.: Финансы и статистика, 1987. – 191 с.

23. Байцер Б. Архитектура вычислительных комплексов. Т. 1. – М.: Мир, 1974. – 498 с.