Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Санкт-Петербургский государственный университет
информационных технологий механики и оптики

Муромцев Дмитрий Ильич

ВВЕДЕНИЕ В ТЕХНОЛОГИЮ ЭКСПЕРТНЫХ СИСТЕМ

УЧЕБНОЕ ПОСОБИЕ

Санкт-Петербург 2005

УДК [004.891 + 002.53:004.89] (075.8)

Д.И. Муромцев. Введение в технологию экспертных систем. – СПб: СПб ГУ ИТМО, 2005. – 93 с.

В учебном пособии рассматриваются основные подходы и методы технологии проектирования экспертных систем. Во введении излагаются базовые вопросы функционирования и этапов создания систем. Вторая глава посвящена методам представления знаний. В третье главе излагаются алгоритмы и математический аппарат логического вывода знаний. В заключительной главе рассматриваются различные подходы к представлению неопределенности в экспертных системах.

Одобрено на заседании кафедры проектирования компьютерных систем Санкт-Петербургского государственного университета информационных технологий механики и оптики.

Рекомендовано учебно-методическим объединением вузов Российской Федерации по образованию в области радиотехники, электроники, биомедицинской техники и автоматизации в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности 220500 «Проектирование и технология электронно-вычислительных средств» направления 654300 «Проектирование и технология электронных средств».

Содержание

1. Введение

В этом разделе раскрывается общее представление об экспертных системах. Кратко изложена история возникновения этого направления в искусственном интеллекте. Описано назначение экспертных систем и их отличие от обычных компьютерных программ. Приведены структура, технология создания и участники разработки экспертных систем. Вводный раздел заканчивается обзором некоторых коммерческих экспертных систем и описанием альтернативных подходов к созданию интеллектуальных систем.

1.1. Думать или вычислять?

В 1950 году английский математик Тьюринг поставил вопрос «могут ли машины думать?». В те времена предположение «да, через 50 лет» Тьюрингу показалось вполне осуществимым.

Стремительное развитие компьютеров и методов программирования, начиная со второй половины 20-го века, постоянно расширяет область применения вычислительной техники. Сейчас уже никого не удивляет стремление заменить человека машиной. Практически во всех областях деятельности, от простейших технологических операций на конвейерном производстве до экспертного анализа и принятия решений, автоматические системы работают не хуже средней руки специалиста. Более того, машина не подвержена так называемому «человеческому фактору» – допускаемыми людьми ошибкам, которые не возможно формализовать и предсказать.

Не исключением стала и интеллектуальная деятельность человека. С момента зарождения кибернетики разработчики компьютерных программ пытались воспроизвести механизм мышления человека или, иначе говоря, ставилась задача научить компьютер «думать». Начало исследованиям в области создания и использования интеллектуальных систем положили работы «отца кибернетики» Норберта Винера [1] и Г.С. Альтшуллера [2].

Первые попытки создания интеллектуальных систем сводились к разработке программ, решающих задачи с помощью разнообразных эвристических методов, основанных на свойственном человеческому мышлению обобщении, использованию универсальных подходов к решению различных задач. То есть усилия были направлены на создание универсальных программ. Результатами этой работы явились такие программы, как ЛОГИК-ТЕОРЕТИК, предназначенная для доказательства теорем в исчислении высказываний, и ОБЩИЙ РЕШАТЕЛЬ ЗАДАЧ,

созданные Ньюэллом, Саймоном и Шоу, занимавшихся исследованием процессов решения различных задач [3]. Также следует отметить всевозможные игровые программы и вычислительные системы.

Так, например, были созданы кибернетические игрушки типа "электронной мыши" Клода Шеннона, которая управлялась сложной релейной схемой. Эта мышка могла "исследовать" лабиринт, и находить выход из него. В последствии, помещенная в уже известный ей лабиринт, она не пыталась искать выход заново, а, используя накопленную информацию, сразу же выходила из лабиринта, не заглядывая в тупиковые ходы.

Американский кибернетик А. Самуэль разработал программу, играющую в шашки. Причем в ходе игры машина обучалась, совершенствуя свою игру на основе накопленного опыта. В 1962 г. эта программа сразилась с Р. Нили, сильнейшим шашистом в США и победила. Такой высокий результат машине удалось достичь благодаря вычислению на каждом шагу игры некоторой оценочной функции, числового показателя, оценивающего качество хода. Эта функция была основана на сочетаниях (в виде линейной комбинации с экспериментально подбираемыми коэффициентами или более сложным образом) знаний о правилах игры, стратегиях и приемах выигрывания (например, как в шашках, и так и в шахматах обычно невыгодно терять свои фигуры, и, напротив, выгодно брать фигуры противника; подвижность фигур и право выбора ходов позволяет держать под боем большое число полей на доске и пр.), а также знаниях, относящихся к отдельным стадиям игры – дебюту, миттэндшпилю, эндшпилю. Сравнивая между собой показатели эффективности различных возможных на данном шаге ходов, машина выберет ход, соответствующий наибольшему показателю. Совершенствование игры состоит в подстройке параметров (коэффициентов) оценочной функции на основе анализа совершенных ходов и игр с учетом их исхода. Следует отметить, что все эти элементы интеллекта заложены в программу ее автором. И хотя машина и совершенствует свою стратегию игры в процессе самообучения, способность выигрывать основана на вычислительной мощности ее процессора. К примеру, компьютер фирмы IBM, победивший в шахматы мирового чемпиона Каспарова, имел 256 процессоров, каждый из которых имел 4 Гб дисковой памяти и 128 Мб оперативной. Весь этот комплекс мог просчитывать более 100 000 000 ходов в секунду.

Еще одним примером является программа американского математика Хао Ванга. Эта программа за 3 минуты работы IBM-704 вывела 220 относительно простых лемм и теорем из фундаментальной математической монографии, а затем за 8.5 минут выдала доказательства еще 130 более сложных теорем, часть их которых еще не была выведена

математиками. Правда, до сих пор ни одна программа не вывела и не доказала ни одной теоремы, которая была бы принципиально новой.

Однако, несмотря на некоторые интересные достижения, попытки создания универсальных программ не привели к существенным открытиям и их промышленному использованию. Разработка таких программ оказалась слишком трудным и, в конечном счете, бесплодным делом. Чем шире класс задач, которые может решать одна программа, тем беднее оказываются ее возможности при решении конкретной частной проблемы.

Дальнейшие исследования в области искусственного интеллекта были сосредоточены не на универсальных алгоритмах решения задач, а на общих методах и приемах программирования, пригодных для создания специализированных программ. Разрабатывались методы представления задачи – способы формулирования проблемы таким образом, чтобы ее можно было легко решить, и методы поиска – эффективные алгоритмы управления ходом решения задачи. Однако значительного продвижения вперед удалось достигнуть в 70-х годах, когда специалисты начали понимать, что эффективность программы при решении задач зависит не только от формализмов и алгоритмов вывода решения, которые она использует, но в первую очередь от знаний, которые в нее заложены. Новая концепция построения интеллектуальных систем привела к развитию специализированных программ со сходной архитектурой, каждая из которых предназначена для решения задач в некоторой узкой предметной области. Эти программы получили название *экспертные системы* (ЭС).

1.2. Назначение экспертных систем

Экспертная система, прежде всего, является программным продуктом, и ее назначение – автоматизация деятельности человека. Однако принципиальным отличием ЭС от других программ является то, что она выступает не в роли «ассистента», выполняющего за человека часть работы, а в роли «компетентного партнера» – эксперта-консультанта в какой-либо конкретной предметной области. ЭС аккумулируют в себе и тиражируют опыт и знания высококвалифицированных специалистов, позволяют пользоваться этими знаниями пользователям «неспециалистам» в данной предметной области. То есть, ЭС не призваны заменить собою эксперта в его непосредственной деятельности, а, напротив, расширяют возможную сферу применения знаний авторитетных специалистов. Кроме того, способности ЭС решать поставленные перед ними задачи не ослабевают со временем и не забываются при отсутствии практики, легко распространяются, так как являются компьютерной программой,

прекрасно документированы, а значит и аргументированы, при многократном решении одной и той же задачи ЭС выдают одно и тоже решение в отличие от человека, который подвержен эмоциональным факторам. Плюс ко всему эксплуатация ЭС значительно дешевле, чем оплата труда человека-эксперта.

Хотя указанные преимущества и очевидны, следует отметить, что ЭС не обладают интуицией и общими знаниями о мире, их ход и метод решения проблемы не может выйти за рамки тех знаний, что в них заложены. ЭС также будут бессильны при решении проблемы в изменяющихся условиях, например, при смене методики решения или появлении нового оборудования. Эксперты могут непосредственно воспринимать весь комплекс входной сенсорной информации, будь то визуальная, звуковая, осязательная или обонятельная. ЭС воспринимает только символы, которыми представлены знания. Поэтому сенсорную информацию необходимо проанализировать и преобразовать в символьную форму, пригодную для машинной обработки. При преобразовании человеком сенсорной информации неизбежно возникают искажения и потери, но классифицировать весь поток информации на значимое и второстепенное или абсурдное способен только человек. Так, например, любой человек сразу же выразит свое недоумение, если его попросят найти номер телефона Аристотеля, но едва ли найдется программа, которая скажет, что древнегреческие философы не пользовались телефонами.

Таким образом, назначением экспертных систем является консультирование по узкоспециальным вопросам при принятии решений человеком. То есть ЭС используются для усиления и расширения профессиональных возможностей их пользователей.

Традиционными областями применения экспертных систем являются следующие [4]:

- *Интерпретация данных*. Это одна из традиционных задач для экспертных систем. Под интерпретацией понимается определение смысла данных, результаты которого должны быть согласованными и корректными. Обычно предусматривается многовариантный анализ данных.
- *Диагностика.* Под диагностикой понимается обнаружение неисправности в некоторой системе. Неисправность – это отклонение от нормы. Такая трактовка позволяет с единых теоретических позиций рассматривать и неисправность оборудования в технических системах, и заболевания живых организмов, и всевозможные природные аномалии. Важной спецификой является необходимость понимания функциональной структуры ("анатомии") диагностирующей системы.
- *Мониторинг*. Основная задача мониторинга – непрерывная интерпретация данных в реальном масштабе времени и сигнализация о

выходе тех или иных параметров за допустимые пределы. Главные проблемы – "пропуск" тревожной ситуации и инверсная задача "ложного" срабатывания. Сложность этих проблем в размытости симптомов тревожных ситуаций и необходимость учета временного контекста.

- *Проектирование*. Проектирование состоит в подготовке спецификаций на создание "объектов" с заранее определенными свойствами. Под спецификацией понимается весь набор необходимых документов чертеж, пояснительная записка и т.д. Основные проблемы здесь – получение четкого структурного описания знаний об объекте и проблема "следа". Для организации эффективного проектирования и, в еще большей степени, перепроектирования необходимо формировать не только сами проектные решения, но и мотивы их принятия. Таким образом, в задачах проектирования тесно связываются два основных процесса, выполняемых в рамках соответствующей ЭС: процесс вывода решения и процесс объяснения.
- *Прогнозирование*. Прогнозирующие системы логически выводят вероятные следствия из заданных ситуаций. В прогнозирующей системе обычно используется параметрическая динамическая модель, в которой значения параметров "подгоняются" под заданную ситуацию. Выводимые из этой модели следствия составляют основу для прогнозов с вероятностными оценками.
- *Планирование*. Под планированием понимается нахождение планов действий, относящихся к объектам, способным выполнять некоторые функции. В таких ЭС используются модели поведения реальных объектов с тем, чтобы логически вывести последствия планируемой деятельности.
- *Обучение*. Системы обучения диагностируют ошибки при изучении какой-либо дисциплины с помощью ЭВМ и подсказывают правильные решения. Они аккумулируют знания о гипотетическом "ученике" и его характерных ошибках, затем в работе способны диагностировать слабости в знаниях обучаемых и находить соответствующие средства для их ликвидации. Кроме того, они способны планировать обучение ученика в зависимости от его успехов.

1.3. Структура и принцип работы экспертных систем

Все ЭС имеют сходную архитектуру. В основе этой архитектуры лежит разделение знаний, заложенных в систему, и алгоритмов их обработки. Так, например, программа решающая квадратное уравнение, несомненно, использует знание о том, как следует решать этот вид

уравнений. Но это знание «зашито» в текст программы и его нельзя не прочитать, не изменить, если исходные тексты программы недоступны. Программы подобного класса весьма удобны для тех, кто решает квадратные уравнения целыми днями. Однако если пользователь хочет решить другой тип уравнения ему не обойтись без программиста, который сможет написать ему новую программу. Теперь, предположим, задача поставлена несколько иначе: программа должна считывать при запуске тип уравнения и способ его решения из текстового файла, и пользователь должен иметь возможность самостоятельно вводить новые способы решения уравнений, например, чтобы сравнить их эффективность, точность и пр. Формат этого файла должен быть одинаково «понятен» как компьютеру, так и пользователю. Такой способ организации программы позволит изменять ее возможности без помощи программиста. Даже если пользователь решает только один тип уравнений новый подход предпочтительней прежнего хотя бы потому, что понять принцип решения уравнений, можно просто изучив входной текстовый файл. Данный пример, несмотря на свою простоту и нетипичность предметной области для применения технологии ЭС (для решения математических уравнений обычно используют специализированные пакеты программ, а не экспертные системы), хорошо иллюстрирует особенность архитектуры ЭС – наличие в ее структуре базы знаний, которую пользователь может просмотреть непосредственно или с помощью специального редактора. Базу знаний можно также редактировать, что позволяет изменять работу ЭС без ее перепрограммирования.

Реальные ЭС могут иметь сложную, разветвленную структуру модулей, но для любой ЭС необходимо наличие следующих основных блоков (Рисунок 1-1. Обобщенная структура ЭС):

1. **БЗ** – база знаний – наиболее ценный компонент ядра ЭС, совокупность знаний о предметной области и способах решения задач, записанная в форме, понятной неспециалистам в программировании: эксперту, пользователю и др. Обычно знания в БЗ записываются в форме, приближенной к естественному языку. Форма записи знаний получила название *язык представления знаний* (ЯПЗ). В различных системах могут использоваться различные ЯПЗ. Параллельно такому "человеческому" представлению БЗ может существовать во внутреннем "машинном" представлении. Преобразование между различными формами представления БЗ должно осуществляться автоматически, так как редактирование БЗ не подразумевает участие программиста-разработчика.
2. **МВ** – машина вывода – блок, моделирующий ход рассуждений эксперта на основании знаний, заложенных в БЗ. Машина вывода является неизменной частью ЭС. Однако большинство реальных

ЭС имеют встроенные средства управлением ходом логического вывода с помощью так называемых метаправил, записываемых в БЗ.

3. **Р** – редактор базы знаний – предназначен для разработчиков ЭС. С помощью этого редактора в БЗ добавляются новые знания или редактируются существующие.
4. **И** – интерфейс пользователя – блок, предназначенный для взаимодействия ЭС с пользователем, через который система запрашивает необходимые для ее работы данные, и выводит результат. Система может иметь «жесткий» интерфейс, ориентированный на определенный способ ввода и вывода информации, или может включать средства проектирования специализированных интерфейсов для более эффективного взаимодействия с пользователем.

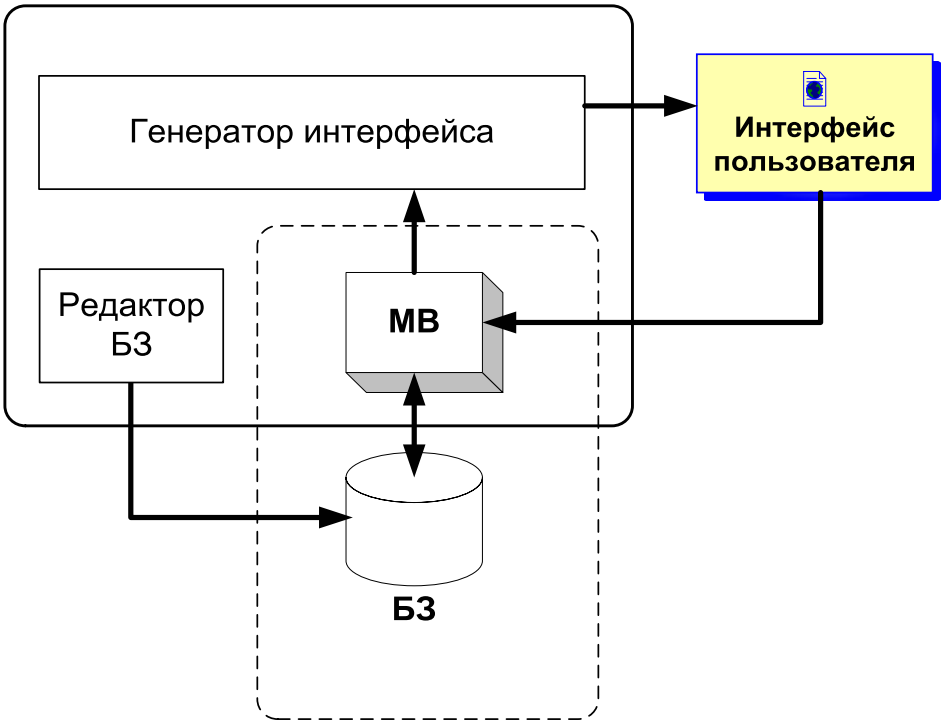


Рисунок 1-1. Обобщенная структура ЭС

С точки зрения изучения технологии экспертных систем наибольший интерес представляют база знаний и машина вывода, различные аспекты реализации которых будут рассмотрены ниже.

В процессе функционирования ЭС считывает информацию из своей базы знаний и пытается осуществить логический вывод решения поставленной перед ней задачи. В базе знаний могут храниться два основных вида записей: факты, описывающие состояние предметной области, составляющие ее объекты и их свойства, а также правила,

описывающие способы решения задачи. Все правила БЗ имеют одинаковую форму записи и состоят из двух частей: условие и действие. Предварительным этапом работы ЭС является сбор исходных фактов, описывающих проблему на языке представления знаний. Эти факты могут поступать в систему различными способами: в режиме диалога через интерфейс пользователя, посредством файлов или баз данных, от внешних датчиков или приборов. После считывания исходной информации машина вывода начинает просмотр базы знаний и последовательно сопоставляет описание задачи с записями БЗ, описывающими ход решения. Если условие текущего правила БЗ подтверждается множеством исходных фактов, то система выполняет действие, записанное в данном правиле, добавляя в БЗ новые, производные факты.

На первый взгляд процесс вывода кажется достаточно простым – выполняются однотипные операции по перебору записей БЗ и сравнении их с имеющимися фактами, пока не будет найдено решение или некий целевой факт. Однако, управление процессом вывода, независящее от контекста проблемы не практике мало эффективно. При решении реальных задач человек крайне редко прибегает к перебору данных. Вместо этого, люди пользуются эвристическими правилами, которые значительно ограничивают пространство поиска решения и позволяет быстро и эффективно решать задачи. Эвристические знания имеют эмпирическую природу, то есть формируются на базе опыта и интуиции эксперта. Ярким примером превосходства эвристического подхода перед алгоритмическим (основанным на полном или частичном переборе) является игра в шахматы [5]. В начале игры «белые» имеют возможность сделать любой из 20 допустимых ходов, в ответ на который «черные» могут также совершить один из 20 ходов. Нетрудно посчитать, что следующий ход «белых» может быть выбран уже из 400 возможных различных состояний партии. Далее, по мере развития игры возникает неуправляемый комбинаторный взрыв. Особенно остро подобная проблема стоит в эндшпиле. Имея по нескольку фигур на доске, каждый из игроков располагает более чем 50 вариантами возможных ходов. Очевидно, шахматные мастера при всем желании не смогли бы осуществлять перебор ходов, для поиска лучшего варианта. Вместо этого они используют краткосрочные и долгосрочные стратегии. Каждая конкретная стратегия выбирается в соответствии с текущей ситуацией на игровой доске.

Другим более простым примером может служить способ строительства стен «сухим» методом. На первом этапе работы имеется большое количество камней различной формы, из которых нужно сложить ровную и устойчивую стену. Более того, камни могут подвозиться по мере необходимости, и осмотр всех камней в целях перебора может быть в принципе невозможен. Строитель вначале не знает, как и какие именно камни он будет выбирать. В процессе строительства время от времени он

осматривает стену, определяет, какие камни остались, и выбирает краткосрочную стратегию, в частности, включающую возврат (удаление камней из стены). Имея один и тот же набор камней, он, возможно, никогда не построит дважды стену одинаково.

Существует два основных типа логического вывода: прямой и обратный. Прямой вывод соответствует обычному ходу решения задачи – от исходных фактов к целевым. Примером прямого вывода является задача классификации. ЭС осуществляет постепенное обобщение исходных фактов, описывающих свойства исследуемого объекта, выявляя наиболее характерные признаки того или иного класса. Обратный вывод соответствует, как следует из названия, обратной задаче – определить какие именно факты требуются для подтверждения данной цели. Этот тип вывода соответствует противоположному ходу решения: сначала машина вывода рассматривает те правила БЗ, действием которых является вывод целевого факта. Затем выбираются новые подцели из условий этих правил, и процесс продолжается от целевых фактов к исходным. Можно сказать, что при обратном выводе происходит конкретизация свойств исследуемого объекта. Этот вид логического вывода наделяет ЭС новым фундаментальным свойством – способностью объяснить, как было получено решение, или что требуется, для того, чтобы имел место тот или иной факт.

В реальных системах, как правило, используется комбинация из прямого и обратного вывода. А для управления всем процессом логического вывода предназначены метаправила – специальный вид правил БЗ, представляющие собой директивы машины вывода. Используя метаправила можно упорядочить применение знаний в зависимости от конкретных значений фактов и текущего состояния БЗ. Продемонстрировать отличие мета правил от обычных правил можно на примере «игрушечной» ЭС. Пусть задачей этой ЭС является размещение мебели (столов, стульев, парт и пр.) в аудиториях университета с учетом требований эргономики, безопасности и т.д. На основании знаний об оборудовании помещения в зависимости от расположения и размеров аудитории, от вида занятий (лекции, практика или лабораторные работы) и других параметров, в БЗ заложены правила предписывающие тот или иной способ размещения мебели. Это обычный вид правил. Но в данной предметной области может понадобиться уточнить способ решения задачи с помощью метаправил вида «Если имеет место свойство X, то сначала применить группу правил N». Таким метаправилом может быть, например, следующее: «Если аудитория предназначена для лабораторных занятий, то сначала применить правила, касающиеся компьютеров и лабораторного оборудования, а затем мебели». Если обычные правила БЗ представляют шаги решения задачи, то метаправила описывают стратегию получения решений.

Тот факт, что фактически изменяемой компонентой в архитектуре ЭС является БЗ, наталкивает на закономерный вопрос: «Можно ли взять готовую экспертную систему из одной предметной области, заложить в нее знания из другой предметной области, и получить новую ЭС?» Для редактирования или даже полной замене содержимого БЗ не требуется изменение кода ЭС и привлечение программистов, поэтому такой перенос готовых программных решений в принципе возможен. Исследования в этом направлении привели к созданию так называемых оболочек экспертных систем. Оболочки ЭС включают машину вывода и интерпретатор ЯПЗ, развитый интерфейс разработчика, а также средства проектирования интерфейса пользователя. Наполнение БЗ оболочки позволяет получить ЭС для различных задач. Повторное использование разработанных компонентов ЭС значительно сокращает время разработки новых ЭС. Однако, как показала практика применения оболочек ЭС, перенос методов решений и средств представления знаний из одной области знаний в другую не всегда возможен. Инструментальные средства, успешно применяемые для одного вида задач, оказываются неэффективными при попытке использовать их для решения других видов задач. Структура и методы описания знаний, в задачах медицинской диагностики и поиска неисправностей в электронных схемах, существенно отличаются от тех, что используются при проектировании технологических цепочек или выборе конфигурации компьютера. Таким образом, возникло новое направление исследований – классификация экспертных задач, таких как медицинская диагностика, планирование, интерпретация сигналов, и т.п. Были предприняты попытки эвристической классификации методов описания знаний и решения проблем в зависимости от решаемой задачи. Такая классификация стала рассматриваться в качестве этапа, предваряющего выбор методов и инструментов решения задач.

1.4. Технология создания экспертных систем

Как уже было отмечено выше, архитектура различных ЭС, с точки зрения входящих в нее программных модулей, идентична практически для любых задач. Детали реализации модулей, конечно, могут сильно отличаются в различных проектах, но их базовый состав и взаимодействие четко определено. Таким образом, при создании ЭС основные усилия должны быть сконцентрированы на проектировании БЗ, в рамках которого выбирается язык представления знаний, способы логического вывода и пр. То есть, несмотря на то, что по своей сути ЭС это программный продукт, разработка новой ЭС сильно отличается от написания новой программы. В

случае же если в качестве инструментального средства используется оболочка ЭС, этап программирования вообще исключается из процедуры создания ЭС.

Учитывая вышесказанное, технологию разработки ЭС можно представить схемой, включающей следующие этапы (Рисунок 1-2. Этапы разработки ЭС.):

1. *Предварительный этап* – этот этап включает деятельность предшествующую решению о разработке новой ЭС. В рамках этого этапа осуществляются конкретизация задачи, подбор экспертов в данной предметной области для совместной работы, выбор подходящих инструментальных средств. Главной особенностью этого этапа является то, что может быть принято решение о нецелесообразности разработки ЭС для выбранной задачи.
2. *Этап прототипирования* – в ходе этого этапа создается прототип ЭС, предназначенный проверки правильности выбранных средств и методов разработки новой ЭС. К прототипу системы не предъявляются высокие требования. Основная его задача состоит в иллюстрации возможностей будущей системы для специалистов, непосредственно участвующих в разработке, а также для потенциальных пользователей. На этом этапе может быть осуществлена корректировка проекта, уточнены время, стоимость и необходимые ресурсы для завершения работы.
3. *Этап доработки* – это по сути основной, наиболее рутинный и продолжительный этап работы над ЭС. Все компоненты многократно тестируются и доводятся до соответствия требованиям проекта. Наибольшую сложность вызывает доработка и доказательство адекватности и эффективности БЗ, так как количество записей в ней может быть на порядок больше, чем в прототипе.

На практике граница между этапами может быть размыта, а сам процесс проектирования является достаточно неформальным, так как связан с исследованием и попыткой копирования деятельности человека. Большое количество применяемых эвристик, интуитивный подход к решению задач экспертами делают процесс создания ЭС творческим. Впрочем, формализация технологии ЭС, разработка в ее рамках математических методов и алгоритмов формирования и обработки знаний – это и есть суть современной теории ЭС. Еще одной особенностью разработки ЭС является поэтапное ее внедрение. Первые версии новой ЭС начинают эксплуатироваться в ограниченном объеме уже на этапе прототипирования.

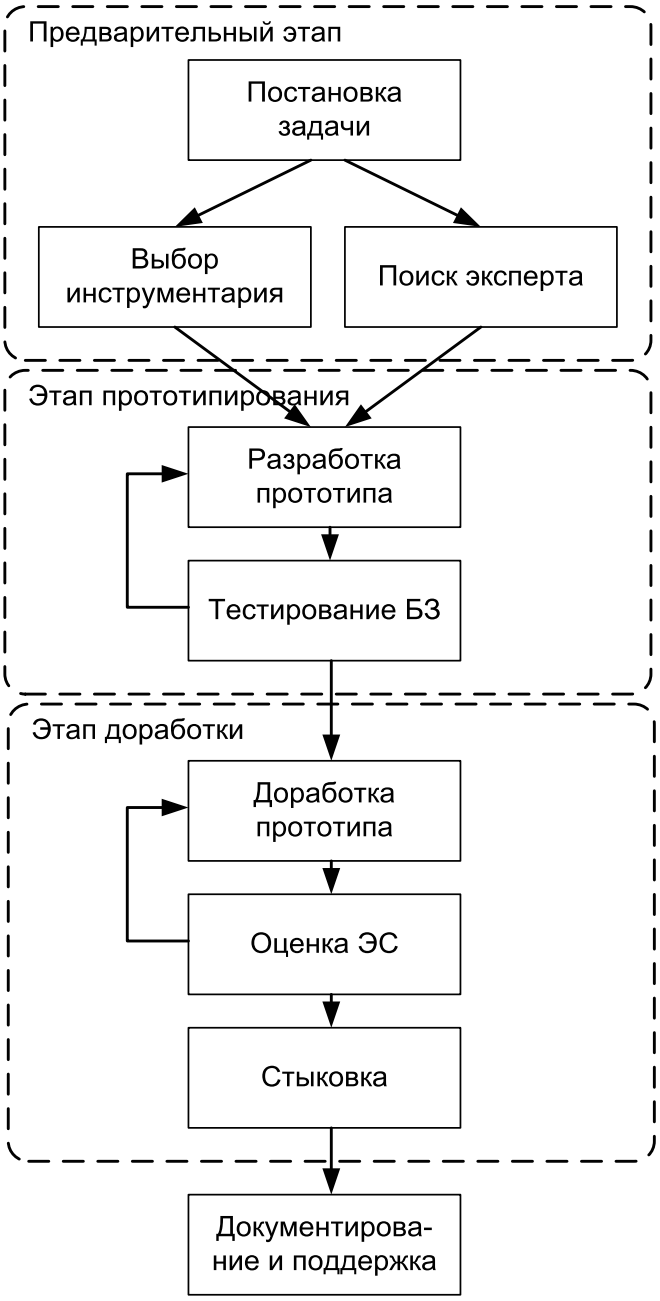


Рисунок 1-2. Этапы разработки ЭС.

Понятие «прототип» требует более подробного рассмотрения. Прототип, как первое приближение к новой ЭС, является индикатором успешности или неудачи предварительного этапа. В процессе постановки задачи наиболее критической частью разработки в целом является правильный выбор проблемы. Если выбрать неподходящую проблему, можно очень быстро увязнуть в «болоте» проектирования задач, которые никто не знает, как решать. Неподходящая проблема может также привести к созданию экспертной системы, которая стоит намного больше, чем экономит, и, по сути, никому не нужна. Дело будет обстоять еще хуже, если разработать систему, которая работает, но не приемлема для

пользователей в силу своей сложности, низкой эффективности или других причин. Если знания, необходимые для решения задач в области применения новой ЭС четко формулируемы и связаны с вычислительной обработкой, то обычные алгоритмические программы, по всей вероятности, будут самым дешевым и эффективным способом решения проблем в этой области.

Экспертная система ни в коем случае не устраняет потребности в базах данных, статистическом программном обеспечении, электронных таблицах и системах текстовой обработки. Но если результативность задачи зависит от субъективного знания, вытекающего частично из соображений здравого смысла или интуиции, тогда эффект от внедрения в данной области экспертной системы может быть достаточно высоким. Критериями, указывающими на необходимость создания ЭС, являются также следующие:

- ✓ нехватка высококвалифицированных специалистов в какой-либо узкой предметной области; система MYCIN была разработана, чтобы консультировать врачей не специализирующихся в области заболеваний крови (примеры ЭС приведены в разделе 1.6).
- ✓ потребность в многочисленном коллективе специалистов, поскольку ни один из них не обладает достаточным знанием; система 1SIS помогала планировать промышленные заказы.
- ✓ сниженная производительность, поскольку задача требует полного анализа сложного набора условий, а обычный специалист не в состоянии просмотреть (за отведенное время) все эти условия; системы FALCON и др. предназначались для контроля аварийных датчиков на химическом заводе.
- ✓ большое расхождение между решениями самых опытных экспертов и новичков; система XCON, разработанная для конфигурирования компьютеров VAX – 1/780/ ошибалась на порядок меньше, по сравнению со специалистом средней руки.

Для того, чтобы знания, необходимые для решения поставленной задачи могли быть успешно представлены в базе знаний, они должны быть достаточно узкоспециализированными, не зависеть в значительной степени от общечеловеческих знаний (знаний о мире или о том, что само собой разумеется). Задачи, решаемые с помощью этих знаний не должны быть для человека-эксперта ни слишком легкими, что сделает применение ЭС равносильным стрельбе из пушки по воробьям, ни слишком сложными, поскольку формализация необходимых знаний может оказаться сверх трудоемкой задачей. Также должен существовать способ каким-либо образом оценить результаты решения поставленной задачи, что позволит судить об адекватности создаваемой ЭС.

Способ, каким ЭС будет решать задачи, зависит от эксперта, знания которого будут заложены в базу знаний. Поэтому немаловажным этапом

является выбор такого эксперта. Некоторые системы могут содержать стратегии одного индивида, а другие представлять подходы коллектива специалистов. Следовательно, выбор подходящего эксперта или экспертов это один из важнейших шагов в создании экспертной системы, так как определяет ее авторитетность, что, в конечном счете, обеспечивает коммерческий успех.

Реализация прототипа возможна с применением специализированных инструментальных средств. На рынке представлен достаточно большой их спектр, начиная от языков программирования (LISP, Prolog, Python и др.), оболочек ЭС (Exsys и пр.) и кончая дорогостоящими комплексами (Gensym G2). Выбор инструментария в основном зависит от имеющихся ресурсов и времени. Распространен также вариант, когда на ранних стадиях для реализации прототипа используют инструмент, позволяющий сократить время разработки (например, оболочку ЭС), а в последствии, для реализации окончательной версии, переходят на более низкоуровневый, но и более эффективный язык программирования.

В процессе создания прототипа будущей ЭС выполняется проверка правильности кодирования фактов, связей и стратегий рассуждения эксперта в БЗ. Также это хороший способ привлечь эксперта к активному участию в разработке экспертной системы и заинтересовать его, так как, как правило, авторитетные эксперты не обладают достаточным временем, чтобы отвлекаться от основной работы для создания ЭС в полном объеме.

Объем первоначальных версий БЗ прототипа обычно составляет несколько десятков записей. Этого вполне достаточно, чтобы смоделировать процесс принятия экспертом отдельных решений по одному или нескольким вопросам. В тоже время разработка этих нескольких десятков правил требует прохождения всего цикла проектирования. Сама процедура создания прототипа имеет итеративный характер. Шаги повторяются, пока не будет получен удовлетворительный результат. Можно условно выделить следующие основные стадии разработки прототипа [4]:

1. *Идентификация проблемы* – знакомство и обучение коллектива разработчиков, а также создание неформальной формулировки проблемы. Уточняется задача, планируется ход разработки прототипа экспертной системы, определяются необходимые ресурсы (время, люди, компьютеры и т.д.), источники знаний (книги, дополнительные эксперты, методики), имеющиеся аналогичные экспертные системы, цели (распространение опыта, автоматизация рутинных действий и др.), классы решаемых задач и т.д.
2. *Извлечение знаний* – процесс взаимодействия эксперта и инженера по знаниям, являющимся основным разработчиком БЗ. В

результате этого взаимодействия инженер по знаниям получает наиболее полное представление о предметной области и способах решения задач в ней. Происходит перенос компетентности экспертов на инженеров по знаниям с использованием различных методов: анализ текстов, диалоги, экспертные игры, лекции, дискуссии, интервью, наблюдение и другие.

3. *Структурирование или концептуализация знаний* – разработка неформального описания знаний о предметной области в виде графа, таблицы, диаграммы или текста, которое отражает основные концепции и взаимосвязи между понятиями предметной области. Выявляется структура полученных знаний о предметной области, т.е. определяются: терминология, список основных понятий и их атрибутов, отношения между понятиями, структура входной и выходной информации, стратегия принятия решений, ограничения стратегий и т.д. Такое описание называется *полем знаний*.
4. *Формализация знаний* – разработка базы знаний на языке, который, с одной стороны, соответствует структуре поля знаний, а с другой – позволяет реализовать прототип системы на следующей стадии программной реализации. Строится формализованное представление концепций предметной области на основе выбранного языка представления знаний (ЯПЗ). Традиционно на этом этапе используются: логические методы (исчисления предикатов I порядка и др.), продукционные модели (с прямым и обратным выводом), семантические сети, фреймы, функциональное программирование, объектно-ориентированные языки, основанные на иерархии классов, объектов и др., а также комбинация указанных методов.
5. *Реализация* – разработка компьютерной программы, демонстрирующей жизнеспособность подхода в целом. Чаще всего первый прототип отбрасывается на этапе реализации действующей ЭС. Создаваемый прототип экспертной системы должен включать базу знаний и все остальные блоки, которые могут быть разработаны с помощью каких-либо языков программирования, оболочек ЭС или специализированных программных комплексов.
6. *Тестирование* – выявление ошибок в подходе и реализации прототипа и выработка рекомендаций по дальнейшей доводке системы до промышленного варианта или повторной переработке. Оценивается и проверяется работа прототипа с целью приведения в соответствие с реальными запросами пользователей. Прототип проверяется на удобство и адекватность интерфейсов ввода-вывода (характер вопросов в диалоге, связность выводимого

текста результата и др.), эффективность стратегии управления выводом на знаниях (порядок перебора, использование нечеткого вывода и др.), качество тестовых примеров, корректность базы знаний (полнота и непротиворечивость правил).

В результате многократного прохождения указанных выше шагов прототип системы постепенно совершенствуется и расширяется. Таким образом, происходит плавный переход от демонстрационного прототипа к коммерческой системе. При этом если программный инструментарий выбран удачно, отпадает необходимость переписывания финальной версии системы на каком-либо языке программирования или другом, более мощном инструментарии.

В плавном процессе совершенствования прототипа системы можно также выделить некие стадии, граница между которыми достаточно размыта, но в тоже время обладающие определенными характеристиками:

- ✓ *Демонстрационный прототип ЭС* – система решает часть задач, демонстрируя жизнеспособность подхода (несколько десятков записей БЗ).
- ✓ *Исследовательский прототип ЭС* – система решает большинство задач, но не устойчива в работе и не полностью проверена (несколько сотен записей БЗ).
- ✓ *Действующий прототип ЭС* – система надежно решает все задачи на реальных примерах, но для сложной задачи, возможно, потребуется много времени и памяти.
- ✓ *Рабочая система* – система обеспечивает высокое качество решений при минимизации требуемого времени и памяти: переписывается с использованием более эффективных инструментальных средств.
- ✓ *Коммерческая система* – рабочая система, пригодная к продаже, т.е. хорошо документирована и снабжена необходимым сервисом по распространению.

Основное на третьем этапе заключается в добавлении большого числа дополнительных эвристик. Эти эвристики обычно увеличивают *глубину* системы, обеспечивая большее число правил для трудноуловимых аспектов отдельных случаев. В то же время эксперт и инженер по знаниям могут расширить охват системы, включая правила, управляющие дополнительными подзадачами или дополнительными аспектами экспертной задачи (метазнания). На этом этапе разработки большинство экспертов узнают достаточно о вводе правил и могут сами вводить в систему новые правила. Таким образом, начинается процесс, во время которого инженер по знаниям передает право собственности и контроля над системой эксперту для уточнения, детальной разработки и обслуживания.

Как только появляется рабочая экспертная система, необходимо провести ее тестирование с точки зрения критериев эффективности, а также оценить степень ее интегрированности со своим окружением (приборами, другими программными продуктами и пр.). К тестированию широко привлекаются другие эксперты для апробации работоспособности системы на различных примерах. Экспертные системы оцениваются главным образом для того, чтобы проверить точность работы программы и ее полезность. Оценку можно проводить, исходя из критериев следующих категорий:

- ✓ критерии пользователей (понятность и «прозрачность» работы системы, удобство интерфейсов и др.);
- ✓ критерии приглашенных экспертов (оценка советов-решений, предлагаемых системой, сравнение ее с собственными решениями, оценка подсистемы объяснений и др.);
- ✓ собственные критерии коллектива разработчиков (эффективность реализации, производительность, время отклика, дизайн, широта охвата предметной области, непротиворечивость БЗ, количество тупиковых ситуаций, когда система не может принять решение, анализ чувствительности программы к незначительным изменениям в представлении знаний, весовых коэффициентах, применяемых в механизмах логического вывода, данных и т.п.).

Отлаженную ЭС необходимо внедрить в ее будущее окружение. Это подразумевает стыковку модулей экспертной системы с другими программными средствами, приборами и пр., используемыми в среде, где будет функционировать новая ЭС. Также необходимо обеспечить обучение пользователей системы. Иногда это означает внесение существенных изменений, которые требуют вмешательства разработчиков системы. Подомными изменениями могут быть, например, изменение интерфейса пользователя, реализация низкоуровневых средств ввода/вывода и пр. Стыковка подразумевает также совершенствование системных факторов, зависящих от времени, вычислительных ресурсов и платформы функционирования ЭС, в целях обеспечения ее более эффективной работы и улучшения технических характеристик, если система работает в неоднородной среде (например, связь с измерительными устройствами).

В качестве примера можно привести стыковку системы PUFF со своим окружением. Эта экспертная система предназначалась для диагностики заболеваний легких. После того, как PUFF была закончена, и все были удовлетворены ее работой, систему перекодировали с LISP на Бейсик. Затем систему перенесли на персональный компьютер, работающий в больнице. В свою очередь, этот компьютер был связан с измерительными приборами. Данные с измерительных приборов сразу же поступали в компьютер, PUFF обрабатывала эти данные и печатала рекомендации для врача. Врач в принципе не взаимодействует с PUFF, так

как система полностью интегрирована со своим окружением – она представляла собой интеллектуальное расширение аппарата исследования легких, который врачи давно использовали.

Другой пример стыковки системы со своим окружением, это система CAT-1 – экспертная система для диагностики неисправностей дизелей локомотивов. Эта система была разработана также на LISP, а затем переведена на FORTH, чтобы ее можно было более эффективно использовать в различных локомотивных цехах. Мастер по ремонту запрашивал систему: определить возможные причины неисправности дизеля. Система связана с видеодиском, с помощью которого мастеру дают визуальные объяснения и подсказки относительно более подробных проверок, которые ему нужно сделать. Кроме того, если оператор не уверен в том, как устранить неисправность, система предоставляла ему обучающие материалы, которые фирма подготовила предварительно. Таким образом, мастер по ремонту может с помощью экспертной системы мог диагностировать проблему, найти нужную тестовую процедуру, получить на дисплее объяснение, как провести тест, или получить инструкции о том, как справиться с возникшей проблемой.

При перекодировании системы на язык, подобный Си, повышается ее быстродействие и увеличивается переносимость, однако гибкость при этом уменьшается. Это приемлемо лишь в том случае, если система сохраняет все знания о проблемной области, и это знание не будет изменяться в ближайшем будущем. Однако, если экспертная система создана именно из-за того, что проблемная область изменяется, то необходимо поддерживать систему в инструментальной среде разработки. Удачным примером ЭС, внедренной таким образом, является XCON (R1) – ЭС, которую фирма DEC использовала для комплектации ЭВМ семейства VAX. Одна из ключевых проблем, с которой столкнулась фирма DEC, – это необходимость постоянного внесения изменений в БЗ для новых версий оборудования, новых спецификаций и т.д. Для этой цели XCON была состыкована со средой проектирования OPS5, в которую поступали все необходимые материалы о новом оборудовании. Таким образом новые данные автоматически попадали в БЗ ЭС, что позволяло практически без дополнительных затрат осуществлять обновление БЗ.

1.5. Команда разработчиков экспертных систем

В разработке любого программного обеспечения участвуют, как правило, несколько специалистов выполняющих различные роли: заказчик, постановщик задачи, аналитик, проектировщик, программист, кодировщик, тестировщик и др. Хотя подобное теоретическое разделение

бывает часто условным, осознание коллективом разработчиков своих функций и задач представляется целесообразным. Роли разработчиков являются постоянными на протяжении всей разработки. Совмещение ролей нежелательно, но на практике, часто по объективным или субъективным причинам, специалисты все же совмещают несколько ролей. Иногда, в силу особенностей проекта, некоторые роли оказываются лишними. В случае простого проекта, опытный специалист может реализовать его в одиночку, взяв на себя все необходимые роли.

Специфика технологии создания ЭС вводит определенное разделение ролей, несколько отличное от общепринятых участников разработки программного обеспечения. Инициатором создания новой ЭС является заказчик. Он же обычно бывает и основным пользователем, поскольку, как уже было отмечено выше, ЭС строится в некоторой узкоспециализированной области знаний и рассчитана на ограниченный круг специалистов. Заказчик (пользователь) может участвовать при разработке интерфейса пользователя, так как система должна быть органично встроена в существующую схему рабочих операций, иначе эффективность от ее использования может быть снижена.

Успех разработки ЭС зависит практически полностью от работы инженера по знаниям и эксперта. Уже во время первоначальных бесед они должны решить, будет ли их сотрудничество успешным или потребуется привлечение других экспертов или инженеров. Это немаловажно, поскольку обе стороны будут работать вместе, по меньшей мере, в течение одного года. В процессе разработки и последующего расширения системы инженер по знаниям помогает эксперту структурировать знания, определять и формализовать понятия и правила, необходимые для решения проблем. Основная ответственность, при этом, ложиться на инженера по знаниям. Он фактически является проектировщиком системы. Основное отличие инженера по знаниям от проектировщика программного обеспечения заключается в том, что в основном его усилия сконцентрированы на базе знаний, а предметом проектирования являются не алгоритмы программы, а знания, которые он получает от эксперта.

Когда экспертная система близка к завершению, инженер по знаниям должен убедиться в том, что эксперты и пользователи знают, как эксплуатировать и обслуживать ее. Пользователи выполняют также роли тестировщиков. Для подтверждения адекватности и полезности системы важно предоставить каждому из пользователей возможность поставить перед ЭС реальные задачи, а затем проследить, как она выполняет эти задачи, и сравнить эти результаты с теми, что получает человек при решении тех же задач.

В случае, если в качестве инструментария используется низкоуровневое средство разработки, то есть какой-либо язык программирования, то необходимо включить в коллектив разработчиков

одного или нескольких программистов. При стыковке системы со своим окружением (существующим оборудованием и программным обеспечением) также понадобиться помощь программиста. Однако, если система разрабатывается с использованием оболочки ЭС, то роль программиста становиться лишней. А инженер по знаниям вполне может самостоятельно кодировать знания посредством редактора БЗ используемой оболочки.

1.6. Примеры коммерческих экспертных систем

В конце данного очень краткого обзора рассмотрим примеры крупномасштабных экспертных систем.

MYCIN – экспертная система, предназначенная для медицинской диагностики заболеваний крови, разработанная группой по инфекционным заболеваниям Стенфордского университета. MYCIN включает базу данных пациентов и базу знаний, состоящую из 450 правил. Для нового пациента в базе данных создается специальная запись, куда помещается информация о симптомах и первоначальных тестов. Система автоматически или при помощи оператора выбирает цель в виде «в крови имеется такой-то микроорганизм», после чего начинается вывод вспомогательных целей, для которых, возможно, потребуются новые анализы. Результаты новых анализов также вводятся в систему, и процесс продолжается, пока система не поставит соответствующий диагноз и выработает рекомендацию по медикаментозному лечению найденной инфекции. В системе реализованы механизмы представления нечетких знаний. Так, запись о пациенте в базе данных представляет собой древовидную структуру, в листьях которой помимо значений параметров хранятся степени уверенности в истинности этих параметров. Правила базы знаний системы также используют степени уверенности при описании знаний. Для вывода заключения в MYCIN используется сначала прямой механизм, в ходе которого выводятся подцели – возможные инфекции, а затем обратный – предназначенный для поиска анализов, способных подтвердить или опровергнуть найденные в ходе прямого вывода подцели. В процессе логического вывода система комбинирует степени уверенности данных и правил, получая, таким образом, степень истинности заключения.

PUFF – анализ нарушения дыхания. Данная система представляет собой MICIN, из которой удалили данные по инфекциям и вставили данные о легочных заболеваниях.

DENDRAL – распознавание химических структур. Данная система старейшая, из имеющих звание экспертных. Первые версии данной системы появились еще в 1965 году во все том же Стенфордском

университете. Пользователь дает системе DENDRAL некоторую информацию о веществе, а также данные спектрометрии (инфракрасной, ядерного магнитного резонанса и масс-спектрометрии), и та в свою очередь выдает диагноз в виде соответствующей химической структуры.

PROSPECTOR – экспертная система, созданная для содействия поиску коммерчески оправданных.

SIAP – обнаружение и идентификация различных типов океанских судов.

АВТАНТЕСТ, МИКРОЛЮШЕР и др. – определение основных свойств личности по результатам психодиагностического тестирования в системах.

ANGY – диагностика и терапия сужения коронарных сосудов.

CRIB и др. – диагностика ошибок в аппаратуре и математическом обеспечении ЭВМ.

СПРИНТ – контроль за работой электростанций.

REACTOR – помощь диспетчерам атомного реактора.

FALCON и др. – контроль аварийных датчиков на химическом заводе.

XCON (или R1) – проектирование конфигураций ЭВМ VAX – 1/780/

CADHELP – проектирование БИС.

SYN и др. – синтез электрических цепей.

WILLARD – предсказание погоды.

PLANT – оценки будущего урожая.

ECON и др. – прогнозы в экономике.

STRIPS – планирование поведения робота.

1SIS – планирование промышленных заказов.

MOLGEN и др. – планирование эксперимента.

УЧИТЕЛЬ ЛИСПА – обучение языку программирования Лисп в системе.

PROUST – обучение языку Паскаль и др.

1.7. Альтернативные подходы к решению интеллектуальных задач

Наряду с технологией экспертных систем в области искусственного интеллекта разрабатываются и другие подходы к решению интеллектуальных задач. Также как и в случае с ЭС эти подходы основаны на имитации умственной деятельности человека. Но, если ЭС – имитируют процесс рассуждения на понятийном уровне, не затрагивая физиологических и психологических процессов, то альтернативные

экспертным системам подходы как раз пытаются воспроизвести механизмы мышления, происходящие в мозгу человека.

Существуют различные способы классификация интеллектуальных технологий. Но наиболее существенным, по всей видимости, является разделение по признаку способности к самообучению. ЭС в классическом варианте просто копируют знания эксперта-человека. Но человеку, прежде всего, присуща способность к обучению. Человеческий мозг постоянно обрабатывает огромное количество сенсорной информации. На основании этой информации, человек вырабатывает определенные реакции, ведущие к достижению текущей цели. Если реакция неверна, то в аналогичной ситуации в будущем, как правило, человек учитывает опыт прошлого и определенным образом изменяет свою реакцию. Существенно, что такая схема поведения верна не только для человека, но и для животных, действующих на уровне рефлексов. Таким образом, интеллектуальное поведение и обучение заложены в самой структуре мозга, обеспечивающей процессы мышления.

Технология создания структур, подобных структуре мозга, получила название нейрокибернетика. Физиологами давно установлено, что основой человеческого мозга является большое количество связанных между собой и нервных клеток – нейронов, взаимно влияющих друг на друга посредством электрических сигналов и способных менять свои характеристики. Поэтому усилия нейрокибернетики были сосредоточены на создании элементов, аналогичных нейронам, и их объединении в функционирующие системы. Эти системы принято называть *нейронными сетями*, или *нейросетями*.

Первые нейросети были созданы в конце 50-х гг. американскими учеными Г. Розенблаттом и П. Мак-Кигюком. Это были попытки создать системы, моделирующие человеческий глаз и его взаимодействие с мозгом. Устройство, созданное ими, получило название *персептрона*. Оно умело различать буквы алфавита, но было чувствительно к их написанию, например, буквы *А*, А и *А* для этого устройства были тремя разными знаками. Но главной особенностью этого устройства была способность к обучению. Перед началом работы устройству демонстрировались обучающие примеры символов, а затем, после завершения обучения, персептрон мог различать различные символы, которым его обучили.

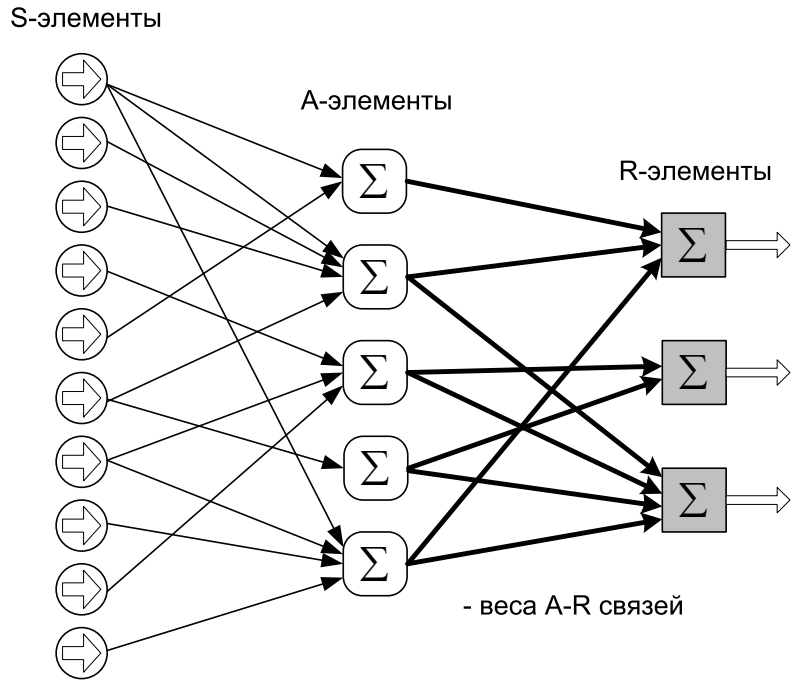


Рисунок 1-3. Структура персептрона.

В наиболее простом виде персептрон (Рисунок 1-3. Структура персептрона.) состоит из совокупности чувствительных (сенсорных) элементов (S-элементов), на которые поступают входные сигналы. S-элементы случайным образом связаны с совокупностью ассоциативных элементов (A-элементов), выход которых отличается от нуля только тогда, когда возбуждено достаточно большое число S-элементов, подающих сигналы на входы A-элемента. A-элементы соединены с реагирующими элементами (R-элементами) связями, коэффициенты усиления (v) которых переменны и изменяются в процессе обучения. Взвешенные комбинации выходов R-элементов составляют реакцию системы, которая указывает на принадлежность распознаваемого объекта определенному образу. Если распознаются только два образа, то в персептроне устанавливается только один R-элемент, который обладает двумя реакциями – положительной и отрицательной. Если образов больше двух, то для каждого образа устанавливают свой R-элемент, выходная реакция которого представляет линейную комбинацию связанных с ним выходов A-элементов:

$$R_j = \Theta_j + \sum_{i=1}^{n} v_{ij} x_i ,$$

где R_j – реакция j-го R-элемента; x_i – реакция i-го A-элемента; v_{ij} – вес связи от i-го A-элемента к j-му R элементу; Θ_j – порог j-го R-элемента. Аналогично записывается уравнение i-го A-элемента:

$$x_i = \Theta_i + \sum_{k=1}^{S} y_k ,$$

Здесь сигнал y_k может быть непрерывным, но чаще всего он принимает только два значения: 0 или 1. Сигналы от S-элементов подаются на входы A-элементов с постоянными весами равными единице, но каждый A-элемент связан только с группой случайно выбранных S-элементов. Предположим, что требуется обучить персептрон различать два образа V_1 и V_2. Будем считать, что в персептроне существует два R-элемента, один из которых предназначен образу V_1, а другой – образу V_2. Персептрон будет обучен правильно, если выход R_1 превышает R_2, когда распознаваемый объект принадлежит образу V_1, и наоборот. Разделение объектов на два образа можно провести и с помощью только одного R-элемента. Тогда объекту образа V_1 должна соответствовать положительная реакция R-элемента, а объектам образа V_2 – отрицательная.

Персептрон обучается путем предъявления обучающей последовательности изображений объектов, принадлежащих образам V_1 и V_2. В процессе обучения изменяются веса v_i A-элементов. В частности, если применяется система подкрепления с коррекцией ошибок, прежде всего учитывается правильность решения, принимаемого персептроном. Если решение правильно, то веса связей всех сработавших A-элементов, ведущих к R-элементу, выдавшему правильное решение, увеличиваются, а веса не сработавших A-элементов остаются неизменными. Можно оставлять неизменными веса сработавших A-элементов, но уменьшать веса не сработавших. В некоторых случаях веса сработавших связей увеличивают, а не сработавших – уменьшают. После процесса обучения персептрон сам, без учителя, начинает классифицировать новые объекты. Если персептрон действует по описанной схеме и в нем допускаются лишь связи, идущие от бинарных S-элементов к A-элементам и от A-элементов к единственному R-элементу, то такой персептрон принято называть элементарным α-персептроном. Обычно классификация задается учителем. Персептрон должен выработать в процессе обучения классификацию, задуманную учителем.

Позднее возникли и другие, более совершенные модели, различающиеся по строению отдельных нейронов, по топологии связей между ними и по алгоритмам обучения. Среди наиболее известных сейчас вариантов нейронных сетей можно назвать сети с обратным распространением ошибки, сети Хопфилда, стохастические нейронные сети. Нейронные сети наиболее успешно применяются в задачах распознавания образов, в том числе сильно зашумленных. Нейросети еще одним замечательным свойством, сближающим их с человеческим мозгом – они способны работать даже при неполной информации об окружающей среде, то есть, как и человек, на вопросы они могут отвечать не только "да" и "нет" но и "не знаю точно, но скорее да".

Довольно большое распространение получил и *эволюционный* или *генетический подход*. Подобно нейронным сетям, генетические алгоритмы

основываются на свойствах биологических систем. В рамках этого подхода самообучение рассматривается как конкуренция внутри популяции между эволюционирующими кандидатами на роль решения задачи. Генетический подход позволяет получать новое решение задач на основе заданного множества исходных посылок. Более того, в процессе работы алгоритма поиска решения формируется цепочка поколений решений, каждое из которых, по крайней мере, не хуже своего предшественника. Таким образом, в процессе поиска система «обучается» и «совершенствует» свои возможности решить поставленную задачу.

Суть этого подхода характеризуют некоторые биологические аналогии, копирующие идеи Дарвина, Менделя и других исследователей, развивавших идеи эволюции: выживание сильнейших особей популяции в процессе ее развития. Одной из первых теоретических разработок в данном направлении стал *генетический алгоритм Холланда*. Этот алгоритм работает на множестве произвольных структур, которыми могут быть логические формулы, высказывания, нейронные сети, или любые другие формализмы. Главное, чтобы для этих структур существовала какая-либо оценочная функция их эффективности. На основании этой функции осуществляется оценка, будет ли этот экземпляр участвовать в следующем поколении решений или нет. Таки образом осуществляется отбор кандидатов на роль решения. Затем с помощью генетических операций, аналогичных трансформациям генотипа в процессе биологического воспроизводства, создается новое поколение решений-кандидатов.

Первоначальный алгоритм включает следующие шаги:

1. Случайным образом создать начальную популяцию из M структур.
2. Вычислить и запомнить для каждой структуры показатель эффективности ее работы. Если их среднее значение достаточно высокое, то остановить эти вычисления и выдать текущую популяцию структур в качестве результата.
3. Для каждой структуры подсчитать вероятность ее выбора $p = \frac{e}{\overline{E}}$, где e – индивидуальный показатель эффективности, а $\overline{E}$ – суммарный показатель для всех M структур в популяции.
4. Создать следующую популяцию структур в соответствии с вычисленной вероятностью выбора, а также применяя генетические операторы.
5. Повторить, начиная со второго шага.

Таким образом, среднее число «потомков» каждой структуры в следующем поколении пропорционально измеренной эффективности при решении стоящей перед системой задачей: выживают наиболее приспособленные.

Генетическими операторами, используемыми на шаге четыре, являются кроссинговер, мутация и инверсия. *Кроссинговер* (перекрест) – это операция, при которой новые структуры получаются посредством случайного взаимозамещения компонентов в начальных структурах. *Мутации*, модифицирующие произвольным образом одну или несколько компонент в структуре, предназначены для введения в популяцию принципиально новой информации. *Инверсия* изменяет характер связей и последовательность компонентов внутри структуры.

Генетический алгоритм позволяет генерировать новые структуры, которые обладают характеристиками, во всяком случае, не хуже, а в большинстве случаев превосходящими существующие.

Еще одним важным свойством генетических алгоритмов является параллельная природа поиска – альтернативные решения, при переходе из поколения в поколение сходятся к точкам оптимума в пространстве поиска решений. В ходе эволюции формируется множество решений, каждое из которых вносит свой независимый вклад в поиск лучшего решения.

Различные методы машинного обучения показали высокую эффективность при решении практических задач. В последнее время наметилась тенденция к объединению этих методов с технологией ЭС, что позволяет создавать высокоэффективные гибридные интеллектуальные системы.

1.8. Вопросы для самопроверки и упражнения

1. Чем экспертная система отличается от обычных программ? Для каких задач имеет смысл создавать экспертные системы, а для каких – нет?

2. Является ли система поиска в сети World Wide Web экспертной? Если нет, то каких именно свойств ей не достает, чтобы являться таковой?

3. Чем гибридные экспертные системы отличаются от систем с классической архитектурой?

4. В чем заключается особенность технологии создания экспертных систем? Что изменяется более всего в новой экспертной системе при постепенном переходе от прототипа к окончательной версии?

5. Кто из команды разработчиков экспертной системы несет наибольшую ответственность за наполнение базы знаний?

6. Можно ли использовать оболочку экспертных систем для создания систем в разных предметных областях?

7. Каким образом можно оценить точность работы экспертной систем? По каким критериям?

8. Существуют ли подходы к построению программ, имитирующих мыслительную деятельность человека, отличные от технологии экспертных систем?

9. Как происходит обучение нейронной сети? Что именно для этого необходимо?

10. На каких операциях основан эволюционный подход к генерации новых структур знаний?

2. Представление знаний

Под *представлением знаний* понимается методика и форма структурированного описания и хранения в памяти вычислительной машины знаний человека-эксперта. Существует множество различных способов представления знаний, и при построении новой экспертной системы может быть выбран один из них, или использоваться сочетание нескольких способов. Во многом, от выбранного способа зависит успех экспертной системы и эффективность ее работы. Каждый из способов обладает своими преимуществами при решении одних задач, и малоэффективен для других. Наиболее важными критериями, пожалуй, можно назвать качество решения, предоставляемого экспертной системой (то есть процент ошибок, скорость получения решения и пр.) и простоту понимания и модификации базы знаний. При выборе метода представления знаний можно руководствоваться общепринятыми для данной предметной области подходами. Так, например, в задачах проектирования распространены табличные описания, в медицине, как правило, постановка диагноза и лечение болезней описываются в свободной форме, с использованием неформальных правил. Решение игровых задач, доказательство теорем может быть описано с помощью языка логики.

Однако не всегда можно однозначно проецировать способ мышления человека во время решения задач на какую-либо определенную модель представления знания. Большинство задач на практике являются комплексными, и для формального описания знаний, необходимых для их решения, требуется использование нескольких различных моделей. Возможно, что в процессе проектирования потребуется изменить форму представления знаний, в целях повышения эффективности экспертной системы, открытости и модифицируемости базы знаний и пр. Определенные ограничения на доступные способы представления знаний накладывает и используемый инструментарий.

Формально *представление* – это множество синтаксических и семантических соглашений, которое делает возможным описание какого-либо предмета. В искусственном интеллекте под предметом понимается состояние некоторой предметной области или среды, а именно объекты среды, их свойства и отношения, которые существуют между ними. *Синтаксис* представления специфицирует набор правил, регламентирующих объединение символов для формирования корректных выражений на данном языке представления. А *семантика* – определяет, как должны интерпретироваться выражения, построенные в соответствии с синтаксическими правилами.

Но ошибочно считать, что представление знаний – это простое кодирование информации. Замена одних символов на другие не решает проблемы неоднозначности, присущей человеческому языку. Так многие задачи, легко решаемые человеком, с трудом реализуются на машине. В качестве примера можно привести фразу, описывающую тривиальную бытовую ситуацию:

«Молоток ударил о графин, и он разбился»

Для человека очевиден ответ на вопрос *«что разбилось?»*. Но чтобы на этот вопрос смогла ответить машина, следует ввести какое-либо правило, связывающее местоимение «*он*» с представленными в данной фразе объектами. Например, можно задать очередность следования предметов во фразе, и считать что второй предмет – *графин* – должен разбиться. Однако, очевидно, что этот подход работает не всегда:

«Графин ударился о камень, и он разбился»

Для человека очевидно, что и в первом, и во втором случае разбиться должен именно графин. Это объясняется тем, что начиная с первых лет жизни люди накапливают *предварительное знание* о мире. Но чрезвычайно трудно такие знания представлять в машинной памяти.

Конечно, существуют и другой тип задач, основанных на применении знаний. Например, задави математической логики. Пусть нужно определить является ли теоремой логики высказываний формула:

$$(\alpha \wedge (\beta \rightarrow \gamma)) \rightarrow ((\delta \vee \alpha) \wedge (\neg\gamma \rightarrow \neg\beta)).$$

Человеку решить это задание довольно трудно, даже если он легко может изложить правила построения формул в логике высказываний. Машина же, напротив, легко справляется с интеллектуальными задачами такого рода.

Различие между приведенными примерами заключается в том, что знания, необходимые для решения задач в логике высказываний, можно выразить в виде нескольких компактных правил. А для правильного понимания фраз типа «X ударил Y, и он разбился» требуется множество знаний об окружающих нас объектах, которое на первый взгляд может показаться бесконечным, и множество исключений вроде пластиковых молотков и бронзовых или каменных ваз.

В данной главе представлено описание наиболее распространенных форм машинной записи знаний или языков представления знаний, применяемых в современных экспертных системах: продукции, логические модели и основные идеи объектно-ориентированного подхода. В начале главы также рассматриваются языки представления знаний, применявшиеся в первых экспертных системах: таблицы операторов, семантические сети и фреймы.

Существенным фактором, определяющим возможности экспертной системы, является механизм представления нечетких знаний. Нечеткие знания широко используются в таких задачах как прогнозирование,

диагностика, распознавание и др., где невозможно со сто процентной уверенностью говорить о результате. Обсуждение различных способов формального описания нечетких знаний приведено в последующих разделах, так как требует более детального рассмотрения.

2.1. Таблицы решений и таблицы операторов

В 60-х годах для представления знаний в БЗ была предложена табличная форма, получившая название *таблицы решений*. Идея этого подхода очень проста и заключается в использовании таблицы в качестве основной формы описания проблемы и способа ее решения. В этой таблице (см. Таблица 2-1. Структура таблицы решений.) предполагается наличие двух основных частей: строк условий $У_i$ и строк действий $Д_i$. Каждое условие и действие соотносится с набором определенных объектов O_i. Если некоторое условие или действие имеет место для данного объекта, то в соответствующую ячейку таблицы заносится 1, в противном случае используется символ 0.

Таблица 2-1. Структура таблицы решений.

	Объекты:			
Условия:	O_1	O_2	…	O_N
$У_1$	1	0	…	1
…	…	…	…	…
$У_M$	1	1	…	0
Действия:				
$Д_1$	0	1	…	1
…	…	…	…	…
$Д_K$	1	0	…	1

В общем случае в ячейках таблицы решений могут использоваться любые символы, отражающие суть условия или действия: значения и диапазоны значений, коды и номера, символьные данные и т.п. В качестве действия может выступать также набор инструкций, подлежащих выполнению в случае выполнения условий.

Структура таблицы тоже может быть модифицирована под конкретную задачу. Например, может добавиться столбец с кодом операции, которую необходимо выполнять для проверки заданного условия. Возможно использование упрощенного варианта таблицы состоящей из пар «условие-действие» без указания каких-либо объектов.

Не представляет труда и расширение таблицы на новые классы объектов – необходимо лишь добавить в конец таблицы недостающие столбцы. А добавление новых строк, представляется несколько более сложным, так как потребуется внести изменения для всех существующих записей (столбцов).

Алгоритм поиска решения по таблице решений тривиален и прост в реализации. Интерпретирующая программа получает на вход конкретизированный набор условий для заданной ситуации и осуществляет последовательный перебор, подыскивая столбцы, где этот набор выполняется. Если такие столбцы находятся, то выполняется действие, указанное для этой ситуации.

Основным достоинством таблиц решений является высокая степень формализации, наглядности процесса принятия решений. Они строятся регулярным образом и могут наращиваться практически до бесконечности, то есть являются универсальным средством решения задач, для которых возможно описание ситуаций с помощью ограниченного набора условий, например, проектирование деталей, представляющих собой поверхности вращения. Однако если различные ситуации характеризуются разными условиями, то таблицы решений становятся сильно разреженными и делают данный подход малоэффективным.

Подход, во многом близкий к описанному выше, был предложен в системе STRIPS (Stanford Research Institute Problem Solver, 1971). Программа предназначалась для решения проблемы формирования плана поведения робота, перемещающего предметы через множество комнат. Текущее состояние окружающей среды – помещений и предметов в них – представляется набором выражений *предикат-аргумент*, которые в совокупности образуют *модель мира*. Общепринятая для ИИ конструкция *предикат-аргумент*, определяется следующим образом:

<предикат> ::= <предикатный символ> (<аргумент$_1$>, …, <аргумент$_n$>).

Различают одноместные или n-местные предикаты. В случае одноместного предиката считается, что аргумент обладает свойством, выраженным предикатным символом. N-местный предикат описывает отношение между объектами, которые заданы аргументами.

Примером ситуации, описываемой с помощью двухместного предиката, может быть нахождение робота в определенной комнате:

at(robot, roomA).

Данный предикат означает, что объект *robot* находится в комнате *roomA*.

Порядок следования аргументов в предикате определяется разработчиком, но должен быть неизменным везде, где этот предикат используется. Можно было бы написать *at(roomA, robot)* и придать этой записи аналогичный смысл, но использовать одновременно в пределах одной базы знаний записи *at(robot, roomA)* и *at(roomA, robot)* нельзя.

С помощью множества предикатов можно описать текущую модель мира, то есть набор конкретных объектов, их свойств и отношений. Например, исходная ситуация может описываться следующим множеством:

$$W^1 = \{ at(robot, roomA), at(box1, roomB), at(box2, roomC) \}.$$

Конечная ситуация также задается множеством предикатов:

$$W^K = \{ at(box1, roomA), at(box2, roomB) \}.$$

Для описания действий, которые может выполнять робот, используются операторы, применяемые к текущей модели мира. Эти операторы позволяют добавлять или изымать некоторые факты из текущей модели. Например, действие

«Переместить робота из комнаты A в комнату B»

в модели W^1 приводит к формированию новой модели W^2. При этом факт *at(robot, roomA)* будет изъят из модели, а *at(robot, roomB)* – добавлен.

Множество допустимых операций, таких как перемещение робота или перенос предметов, кодируются в таблице операторов (см. Таблица 2-2. Пример таблицы операторов.), которая близка по структуре к таблице решений, но предполагает обязательное наличие модели мира.

Таблица 2-2. Пример таблицы операторов.

Оператор:	*move(X, Y)*	*push(X, Y, Z)*
Предварительные условия:	*{ at(robot, X) }*	*{ at(robot, Y), at(X, Y) }*
Список удалений:	*{ at(robot, X) }*	*{ at(robot, Y), at(X, Y) }*
Список добавлений:	*{ at(robot, Y) }*	*{ at(robot, Z), at(X, Z) }*

Как уже было отмечено, задачей системы STRIPS является формирование плана действий робота для достижения цели. Таким образом, результатом работы системы должна быть последовательность операторов, применение которых к исходной модели мира позволяет достичь целевой модели. Зная целевое состояние среды можно было бы перебирать последовательно или случайно комбинации операторов, пока цель не будет достигнута. Но экспоненциальный рост количества вариантов при каждой новой проверке делает такой подход неприемлемым на практике.

Для предотвращения экспоненциального роста вариантов возможных решений в качестве основы для работы системы был предложен метод «анализ целей и средств», идея которого состоит в том, чтобы с каждой новой операцией отличие между текущим состоянием и целевым уменьшалось. Это предполагает наличие меры оценки «расстояния» в пространстве решений. Например, если очередная подцель сформулирована в виде предиката:

at(box1, roomA),

а ящик *box1* находится в комнате *roomB*, то перемещение робота из комнаты *roomA* в комнату *roomC* не приблизит текущее состояние к целевому (для модели W^1). А перемещение из комнаты *roomA* в комнату *roomB*, наоборот, уменьшит расстояние между текущим и целевым состоянием, так как позволит на следующем шаге переместить ящик *box1* в комнату *roomA*.

Алгоритм поиска требуемых операторов системы STRIPS основан на сопоставлении очередной подцели и списков добавления в таблице операторов. Новые подцели выбираются из списка предварительного условия найденного оператора. Так, например, цель *at(box1, roomA)* соответствует элементу *at(X, Z)* в списке добавлений оператора *push(X, Y, Z)*. Сопоставление этих двух предикатов (*X* соответствует *box1*, а *Z* – *roomA*) позволяет выбрать из предварительного условия оператора новые подцели:

at(robot, Y), at(box1, Y).

Далее необходимо найти в модели мира предикат, содержащий объект (в данном случае это комната), который можно сопоставить с символом *Y*. Таким предикатом может быть, например, *at(box1, roomB)*. Поставив в соответствие с символом *Y* объект *roomB* можно окончательно сформулировать очередные подцели:

at(robot, roomB) и *at(box1, roomB).*

Теперь первый элемент в этом списке указывает желаемое (целевое) положение робота, а второй элемент уже присутствует в модели мира W^1. Следовательно, после применения оператора *push* к модели мира W^1, необходимо добавить *at(robot, roomB)* и удалить *at(robot, roomA)*. В результате получится новая модель:

W^2 = { *at(robot, roomB), at(box1, roomB), at(box2, roomC)* }.

Описанная процедура повторяется до тех пор, пока очередная модель не будет соответствовать целевой.

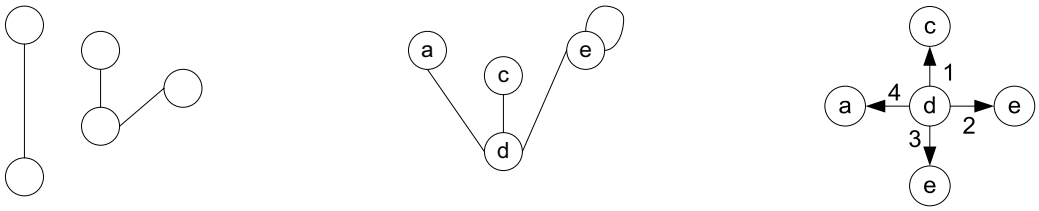
Так как таблица операторов, модель мира и цели представлены с помощью одного и того же синтаксиса в виде конструкций *предикат-аргумент*, то, применяя описанную выше схему сопоставления, программа довольно просто находит, какие именно операции нужно выполнить для достижения поставленной цели. Всё, что нужно для этого сделать, – просмотреть списки добавлений в описании операторов и найти в них элемент, соответствующий заданной цели.

2.2. Семантические сети

Как следует из названия *семантическая сеть*, этот метод представления знаний позволяет описывать объекты, явления и понятия

предметной области с помощью теории графов (*семантика* – это наука, устанавливающая отношения между символами и объектами, которые они обозначают, или наука, определяющая смысл знаков; *сеть* – разновидность графа). Семантические сети первоначально были разработаны для использования их в качестве психологических моделей человеческой памяти, но в последствии с успехом стали применяться в экспертных системах.

Терминология теории графов позволяет описывать практически любые структуры абстрактных данных. Коротко суть этой теории можно сформулировать следующим образом. Для построения графа используются два основных вида примитивов: узлы и связи. Узлы и связи могут быть дополнительно промаркированы. Связи могут быть также направленными и нет. Несколько примеров абстрактных графов представлены ниже (см. Рисунок 2-1. Примеры графов.): а – обыкновенный граф с маркированными вершинами, б – связный граф с петлей, в – простое, ориентированное дерево с маркированными и направленными связями.



а. **б.** **в.**

Рисунок 2-1. Примеры графов.

Семантическая сеть представляет собой ориентированный граф, где узлы представляют понятия предметной области, а связи – отношения между понятиями. В качестве *понятий* обычно выступают абстрактные или конкретные объекты, концепты или события, а *отношения* – это связи следующих видов:

1. связи, определяющие тип объектов ("это есть" или "класс-подкласс", "иметь частью" или "часть-целое", "принадлежать" или "элемент-множество" и т.п.);
2. функциональные связи (определяемые обычно глаголами "производит", "влияет" …);
3. количественные ("больше", "меньше", "равно" …);
4. пространственные ("далеко от", "близко от", "за", "под", "над" ...);
5. временные ("раньше", "позже", "в течение" …);
6. атрибутивные связи (*иметь свойство*, иметь *значение...)*;
7. логические связи ("и", "или", "не") и др.

Характерной особенностью семантических сетей является обязательное наличие в одной сети трех типов отношений:

- ✓ класс - элемент класса;

- ✓ свойство - значение;
- ✓ экземпляр элемента класса.

Проблема поиска решения в базе знаний типа семантической сети сводится к задаче поиска фрагмента сети, соответствующего некоторой подсети, отвечающей поставленному вопросу. Подобного рода задачи решаются с помощью аппарата теории графов. Следует особо отметить роль фундаментальных признаков связей (рефлексивность, симметричность и транзитивность) в процессе вывода на сети. Так, например, связи вида "это есть" или "иметь частью" транзитивны, что позволяет говорить об установлении с помощью этой связи свойств иерархии наследования в сети. Это означает, что элементы более низкого уровня в сети могут наследовать свойства элементов более высокого уровня в сети.

Примером простой семантической сети является описание объекта автомобиль и рядя связанных с ним понятий (см. Рисунок 2-2. Пример простейшей семантической сети.). На этой сети присутствует следующая цепочка понятий: «автомобиль имеет частью двигатель», «двигатель имеет частью стартер». В силу транзитивности отношения "иметь частью" можно вывести следующее утверждение «автомобиль имеет частью стартер». Аналогично можно сделать вполне очевидный вывод, что «Иванов обладает автомобилем» или, что «Mercedes имеет частью двигатель и потребляет топливо».

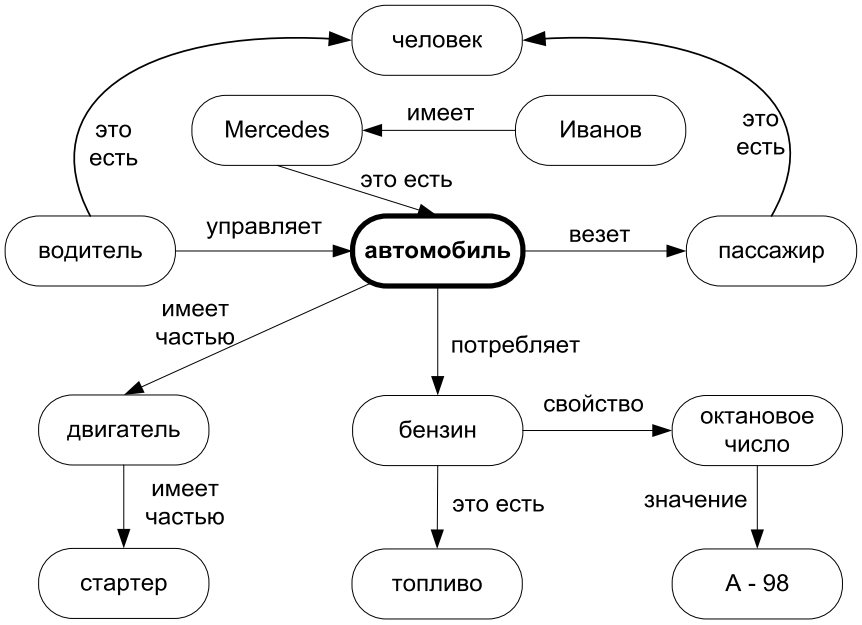


Рисунок 2-2. Пример простейшей семантической сети.

Основное преимущество этой модели – в соответствии современным представлениям об организации долговременной памяти человека и

высокой наглядности представления знаний. В современной практике подобная форма представления получила развитие в самых различных приложениях. Недостатком данной модели является сложность поиска решения и вывода на семантической сети.

Разновидностью семантических сетей являются *ассоциативные сети*, использующие разделение узлов на *узлы-типы* и *узлы-лексемы*. Узлы-типы представляют собственно концепты, а узлы-лексемы определяют понятие типа. Таким образом, определение одних концептов связано (как в толковом словаре) с определением других концептов. Например, смысл слова «автомобиль» можно определить как «машину», являющуюся конструкцией из связанных компонентов, выполняющих некоторую работу. Это определение требует связать тип «машина» с лексемами «конструкция» и «компонент». Если теперь определить тип «компьютер», как разновидность «машины», то можно будет сказать, что компьютер конструкцией из компонентов, выполняющих определенную работу.

2.3. Фреймы

Впервые термин «фрейм» был предложен в 70-е годы Марвином Минским [6], который определил его следующим образом:

> «*Фрейм* – это структура данных, представляющая стереотипную ситуацию, вроде нахождения внутри некоторого рода жилой комнаты, или сбора на вечеринку по поводу дня рождения ребенка. К каждому фрейму присоединяется несколько видов информации. Часть этой информации – о том, как использовать фрейм. Часть о том, чего можно ожидать далее. Часть о том, что следует делать, если эти ожидания не подтвердятся».

По своей организации фрейм похож на семантическую сеть, так как является моделью организации памяти человека. Но, в сравнении с сетью, фрейм представляет собой более естественную форму для отображения иерархически организованных знаний. Такого рода знания базируются на понятии абстрактного образа или ситуации. В психологии и философии известно понятие абстрактного образа. Например, слово "комната" вызывает у слушающих образ комнаты: "жилое помещение с четырьмя стенами, полом, потолком, окнами и дверью, площадью 6-20 $м^2$". Из этого описания ничего нельзя убрать, например, комната без окон – это чулан или кладовка, а не комната. Но абстрактный образ имеет "дырки" в своем определении, – это незаполненные значения некоторых атрибутов – количество окон, цвет стен, высота потолка, покрытие пола и др.

Существует предположение, что представление понятий в мозге человека не требует четкого определения набора свойств, а базируется на понятии типа или класса, объединяющего в себе наиболее общие свойства. Сущности, соответствующие классам получили название *прототипа*. Прототип – это уже не абстрактный образ, а наиболее типичный представитель своего класса, с обобщенными, но вполне конкретными, значениями своих свойств. Например, прототип понятия четырехугольник можно определить как фигуру, имеющую четыре угла. Возникает желание задать значения величин углов «по умолчанию» одинаковыми. И этому есть вполне убедительное объяснение. Рассматривая различные примеры фигур (см. Рисунок 2-3. "Менее" и "более" типичные четырехугольники) слева направо, кажется, что «типичность» объектов увеличивается. Четырехугольник, не обладающий выпуклостью, кажется менее типичным чем прямоугольник, возможно потому, что понятие площадь фигуры в нашем сознании коррелирует с длиной периметра. А это лучше просматривается у фигуры с равными внутренним углами.

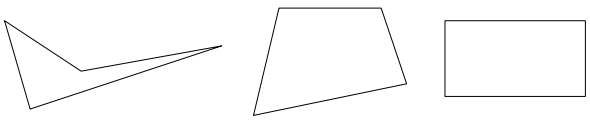


Рисунок 2-3. "Менее" и "более" типичные четырехугольники.

Прототип может быть основой для описания любой сущности данного класса в случае неполноты информации, так как представляет обобщенные свойства большинства разновидностей объектов класса. Так, если требуется оценить площадь более или менее прямоугольного участка земли, для которого известны длины сторон, то можно оценить площадь, предполагая, что внутренние углы его контура равны. В худшем случае, если предположение о равенстве углов будет нарушено, то значение площади будет завышено, что в большинстве случаев вполне допустимо.

Основная идея фреймового подхода заключается в сосредоточении всей информации, относящейся к одному объекту в одной структуре данных, вместо распределения ее по базе знаний. Структура фрейма достаточно проста (см. Таблица 2-3. Структура фрейма.): с представляемым понятием связывается список именованных атрибутов, называемых *слотами*. Каждый слот может иметь значение по умолчанию. Также со слотом могут быть связаны произвольные процедуры, выполняемые при смене его значения. Эти процедуры в литературе иногда именуются «демонами». Со слотом можно связать любое количество процедур, но наиболее часто используются следующие:

1. Процедура на событие «если добавлено» (IF-REMOVED). Выполняется, когда новая информация записывается в слот.

2. Процедура на событие «если удалено» (IF-ADDED). Выполняется, когда информация удаляется из слота.
3. Процедура на событие «по требованию» (IF-NEEDED). Выполняется, когда запрашивается информация из пустого слота.

Таблица 2-3. Структура фрейма.

Название понятия (имя фрейма)		
Слоты:	Значения:	Список связанных процедур:
Имя слота 1	Значение 1	Процедура 1
Имя слота 2	Значение 2	Процедура 2
…	…	…
Имя слота N	Значение N	Процедура N

Фреймы-прототипы, хранящиеся в базе знаний, позволяют реализовать наследование свойств одних фреймов от других. Аналогично тому, как в семантической сети наследование происходит по транзитивным отношениям («имеет частью», «это есть»), фреймы-экземпляры могут наследовать свойства фреймов-прототипов через специальные слоты АКО (*A-Kind-Of* = «это есть»). Слот АКО указывает на фрейм более высокого уровня иерархии, откуда неявно наследуются, то есть переносятся, список и значения слотов. Фреймы-экземпляры создаются в БЗ для отображения реальных объектов и ситуаций на основе поступающих данных. Возможно наследование свойств от нескольких прототипов. Такой вид наследования получи название «множественное наследование».

С помощью фреймовой модели можно представить все многообразие знаний о мире через: *фреймы-структуры*, для обозначения объектов и понятий (устройство, отчет, заказ); *фреймы-роли* (менеджер, пользователь, клиент); *фреймы-сценарии* (банкротство, собрание акционеров, тестирование); *фреймы-ситуации* (тревога, авария, рабочий режим устройства) и др.

В качестве примера можно рассмотреть формирование понятия заказ товара (см. Рисунок 2-4. Пример описания знаний с помощью фреймов.).

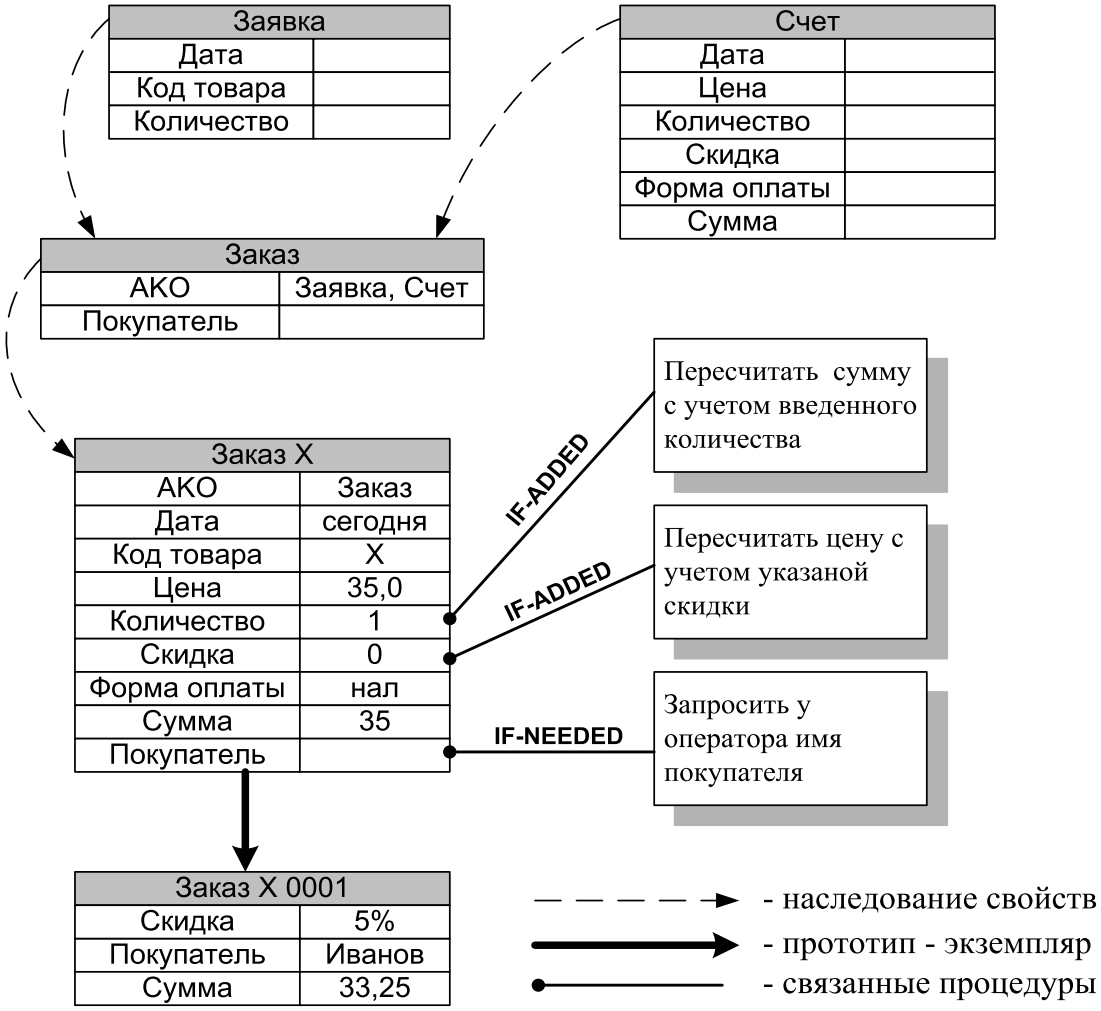


Рисунок 2-4. Пример описания знаний с помощью фреймов.

2.4. Объектно-ориентированный программирование

Объектно-ориентированная технология программирования (ООП) сформировалась в начале 80-х годов. Сегодня она по праву получила широкое распространение для решения самых различных задач: традиционные и интеллектуальные задачи программирования, хранение данных и интерактивная графика, и многое другое. В каком то смысле объектный подход можно считать развитием фреймового представления. Однако если фреймы изначально предназначены для представления знаний и подразумевают решение интеллектуальных задач, то объектно-ориентированная технология является универсальным средством программирования и может применяться для решения любых задач.

Объектно-ориентированному программированию посвящено огромное количество публикаций, которые, в основном, предлагают описание каких-либо коммерческих инструментальных средств. И хотя

большинство из этих инструментариев имеют свои ограничения на объектную модель, подробное ознакомление с сутью подхода можно получить на примере любого объектно-ориентированного языка программирования. В силу указанных соображений, содержание данного раздела ограничено только описанием основных понятий «чистого» объектно-ориентированного подхода.

Представление концептуальных понятий в объектно-ориентированном языке осуществляется с помощью *классов*, задача которых – объединить связанные данные и поведение. Эти классы могут *наследовать* некоторые или все свойства своих родителей – *суперклассов*, они также могут определять свои собственные *атрибуты* (данные) и *методы* (поведение). Классы, как правило, выступают в качестве шаблонов (аналогично прототипам для фреймов) для создания экземпляров классов – *объектов*. Различные экземпляры одного и того же класса обычно содержат различные данные, но имеют единообразное представление, то есть значения атрибутов объектов различны, а их перечень и методы для всех объектов одного класса одни и те же.

При использовании объектно-ориентированного подхода не принято использовать прямой доступ к атрибутам какого-либо класса из методов других классов. Взаимодействие между классами осуществляется с помощью отправления *сообщений*, или вызовов методов объектов. Когда объект получает сообщение, он может по-разному реагировать на него, в зависимости от своего текущего состояния. Методы и атрибуты класса или *члены* класса подразделяются на *открытые* и *закрытые*. Открытые члены класса образуют его *интерфейс*, через который объекты данного класса взаимодействуют с другими объектами. Закрытые члены класса не доступны для других объектов. Сокрытие реализации класса и отделение его внутреннего представления от внешнего получило название *инкапсуляция*.

Еще одним фундаментальным понятием в объектно-ориентированном программировании, наряду с наследованием и инкапсуляцией, является *полиморфизм*. Слово полиморфизм греческого происхождения и означает "имеющий много форм". Формально, полиморфизм – это положение теории ООП, согласно которому одинаковые имена методов могут обозначать различное поведение объектов для нескольких классов, имеющих общий суперкласс. Следовательно, любой объект, обозначаемый полиморфным именем, может по-своему реагировать на некий общий набор операций. Суть изложенного выше заключается в переопределении поведения дочерних классов, в зависимости от их семантики. Примером полиморфизма может быть практически любая иерархия. В зоологии весь животный мир объединен в иерархию видов. Каждое животное способно перемещаться в пространстве, но одни ходят, другие – плавают, третьи – летают, а кто-то

может и летать, и плавать, и ходить. Этот пример как нельзя лучше иллюстрирует полиморфизм применительно к понятию «перемещаться в пространстве».

Некоторые объектно-ориентированные языки программирования предусматривают *интроспективные* (или *рефлексивные*) объекты. Это объекты, в поведении которых предусмотрено предоставления собственного описания: принадлежность к классу данного экземпляра, имена суперклассов, список доступных методов и атрибутов. Интроспекция позволяет функции или методу, управляющему объектами, принимать решения, основываясь на том, какой вид или тип объекта ему передается.

Во многих ООП-языках сами классы также являются объектами, которые можно создавать, передавать и подвергать интроспекции. Но поскольку объекты, как отмечалось, создаются с использованием классов в качестве шаблонов, то для создания объектов-классов также нужны шаблоны. Подобные классы для классов получили название *метаклассы*, которые позволяют динамически генерировать новые классы с различными атрибутами и методами, или, иначе говоря, управлять процессом создания не только объектов, но и классов. Этот аспект особенно важен тогда, когда за ранее невозможно предусмотреть в тексте программы все возможные классы. Примером такой ситуации является анализ текста – концепции формируются по ходу анализа и зависят от содержания нового текста. Наконец, поскольку метакласс сам является классом, то нет никакого смысла в заведении "мета-мета-классов".

Объектно-ориентированное программирование позволяет создавать расширяемые системы. Это одно из самых значительных достоинств ООП, и именно оно отличает данный подход от традиционных методов программирования. Расширяемость означает, что существующую систему можно заставить работать с новыми компонентами, причем без внесения в нее каких-либо изменений. Компоненты могут быть добавлены на этапе исполнения программы. Независимые от предметной области части системы могут быть реализованы в виде набора универсальных классов и в дальнейшем расширены за счет добавления частей, специфичных для конкретного приложения. Эти преимущества делают ООП мощным средством не только представления данных, но и знаний. Однако чтобы эффективно использовать эти преимущества в задачах искусственного интеллекта, требуются значительные вычислительные ресурсы и развитые (как правило, дорогие) инструментальные комплексы.

2.5. Продукционные правила

Системы представления знаний, использующие выражения вида «ЕСЛИ *условие*, ТО *действие*», получили название системы продукций или системы, основанные на правилах. Идея этого метода принадлежит Э. Посту (1943). Правила обеспечивают формальный способ представления рекомендаций, указаний или стратегий. Они идеально подходят в тех случаях, когда знания предметной области возникают из эмпирических ассоциаций, накопленных за годы работы по решению задач в данной области. Представления знаний в виде продукций наиболее распространено в экспертных системах, так как запись знаний фактически ведется на подмножестве естественного языка. Следствием этого является то, что правила легко читаются, их просто понять и модифицировать, эксперты без труда могут сформулировать новое правило или указать на ошибочность какого-либо существующего.

В качестве *условия* и *действия* в правилах может быть, например, предположение о наличии того или иного свойства, принимающее значение *истина* или *ложь*. При этом термин *действие* следует трактовать широко: это может быть как директива к выполнению какой-либо операции, рекомендация, или модификация базы знаний – предположение о наличии какого-либо производного свойства. Примером продукции может служить следующее выражение:

П1: *ЕСЛИ* клиент работает на одном месте более двух лет,
ТО клиент имеет постоянную работу.

Как условие, так и действие правила могут учитывать несколько выражений, объединенных логическими связками *И*, *ИЛИ*, *НЕ*:

П2: *ЕСЛИ* клиент имеет постоянную работу
И клиенту более 18 лет
И клиент НЕ имеет финансовых обязательств,
ТО клиент может претендовать на получение кредита.

Помимо продукционных правил база знаний должна включать и простые *факты*, поступающие в систему через интерфейс пользователя или выводимые в процессе поиска решения задачи. Факты являются простыми утверждениями типа «клиент работает на одном месте более двух лет». И когда в процессе интерпретации правил машиной вывода какой-либо факт согласуется с частью правила *ЕСЛИ*, то выполняется действие, определяемое частью *ТО* этого правила. Новые факты, добавляемые в базу знаний в результате действий, описанных в правилах, также могут быть использованы для сопоставления с частями *ЕСЛИ* других правил. Последовательное сопоставление частей правил *ЕСЛИ* с

фактами порождает *цепочку вывода*. Цепочка вывода, полученная в результате последовательного выполнения правил П1 и П2 показана ниже (см. Рисунок 2-5. Пример цепочки вывода.). Эта цепочка показывает, как на основании правил и исходных фактов выводит заключение о возможности получения кредита. Цепочки вывода экспертной системы могут быть предъявлены пользователю и помогают понять, как было получено решение.

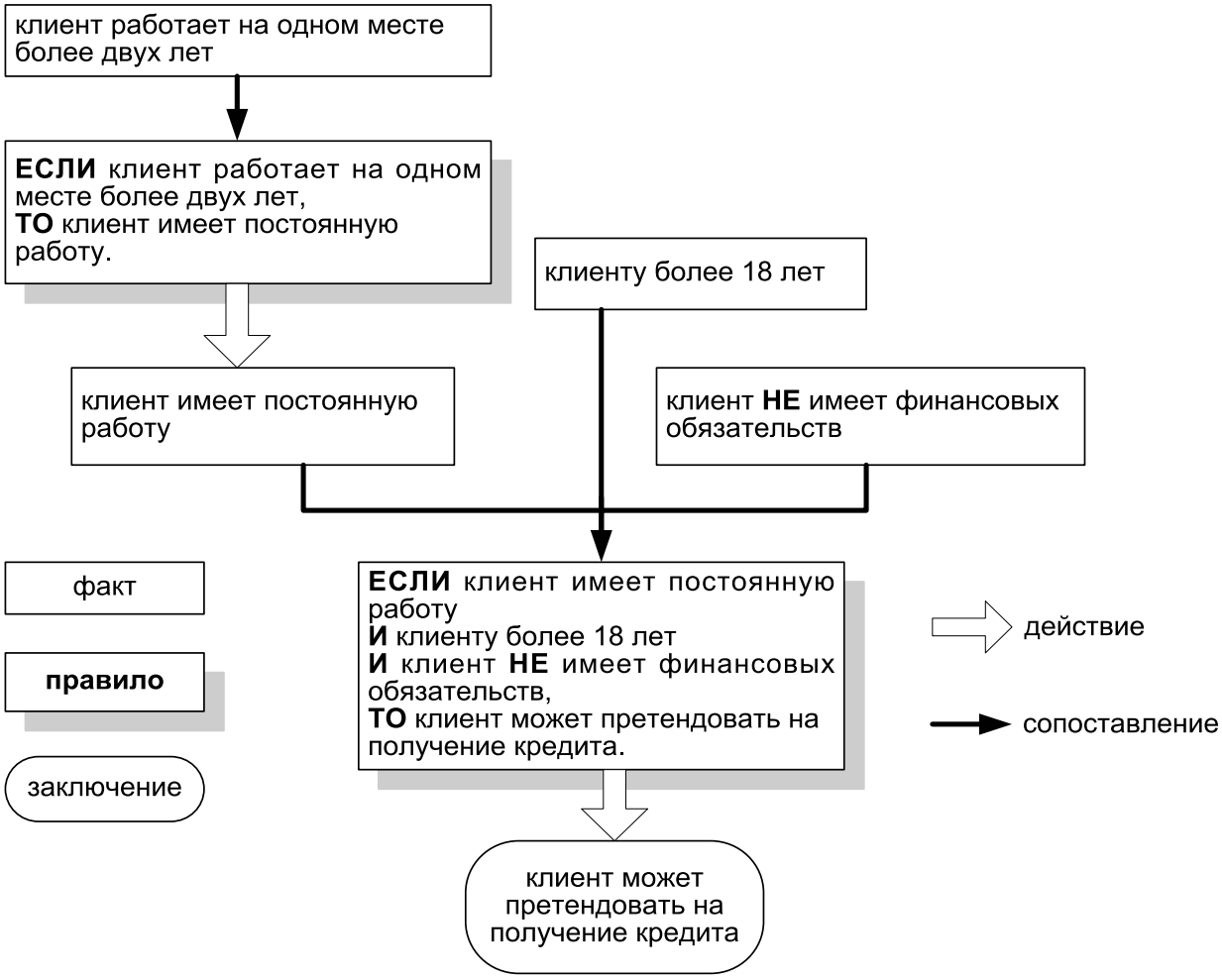


Рисунок 2-5. Пример цепочки вывода.

Существуют два основных способа выполнения правил в системе: *прямая цепочка рассуждений* или *прямой вывод* и *обратная цепочка рассуждений* или *обратный вывод*. В приведенном выше примере использовалась прямая цепочка рассуждений – от исходных фактов к цели (заключению). Этот вид вывода применим в задачах, где на основании имеющихся фактов необходимо определить тип (класс) объекта или явления, выдать рекомендацию, определить диагноз и т.п. Если же задача

формулируется иначе – для заданной ситуации необходимо определить условия к ней приводящие, то используется обратный вывод. Подобная задача может формулироваться, например, следующим образом: «Определить условия, при которых клиент может получить кредит». Обратная цепочка применима также, если требуется объяснить, как было получено решение.

В процессе вывода решения все правила системы равнозначны и самодостаточны, то есть все необходимое для активизации правила содержится в его условии, и одни правила не могут непосредственно вызывать другие. Но иногда для получения решения требуется вмешательство в стандартный процесс вывода. Для этих целей некоторые продукционные системы позволяют вводить в базу знаний специальные правила для управления процессом вывода – *метаправила*. Метаправила не принимают непосредственного участия в процессе формирования рассуждений, а определяют приоритет выполнения или исключают из рассмотрения обычные правила и выполняются в первую очередь. Ниже приведен пример метаправила, сокращающего цепочку вывода:

МП1: *ЕСЛИ* кредитный рейтинг клиента высокий
И клиент является клиентом банка,
ТО сначала применить правила для льготных условий предоставления кредита.

Можно также сформулировать пример метаправила, касающегося общей стратегии вывода и не связанного с какой-либо конкретной предметной областью:

МП2: *ЕСЛИ* существуют правила, в условиях которых не упоминается текущая цель
И существуют правила, в условиях которых упоминается текущая цель,
ТО сначала следует активизировать первые из перечисленных правил.

Последний пример не связан с предметной областью и, поэтому, подобные метаправила можно применять в системах различного назначения. Интерес к такого рода обобщенным формулировкам знаний достаточно высок. Идея использования метаправил является весьма продуктивной, но, тем не менее, метаправилами следует пользоваться осмотрительно, учитывая возможные исключительные ситуации.

Продукционные правила обеспечивают естественный способ описания процессов, управляемый сложной и изменяющейся средой. Через правила можно описать ход решения задачи, не имея заранее алгоритма этого решения. Более того, можно корректировать способ решения,

добавляя новые правила, не изменяя существующих, что обеспечивает высокую модульность базы знаний.

Однако, несмотря на то, что с помощью продукционных правил можно представить решение любой задачи, при большом количестве правил становится сложно отслеживать непротиворечивость базы знаний.

2.6. Логические модели

Логические модели представления знаний основаны на стиле программирования, разработанном Ковальским, который использует логику предикатов для управления анализом декларативных предложений. Каждое предложение записано в следующей форме:

Консеквент → антецедент-1, антецедент-2, … антецедент-n

Данная форма сходна с продукционными правилами. Здесь антецеденты (заключение) являются предикатами, истинность которых можно подтвердить или опровергнуть, а консеквент (условие) – это предикат, который является истинным, если можно доказать истинность всех его антецедентов. Те есть приведенный пример предложения аналогичен продукции «*ЕСЛИ* условие-1 *И* условие-2 *И* … *И* условие-n, *ТО* заключение».

Программа, основанная на логике, берёт поставленную цель и сравнивает её с консеквентами всех запомненных предложений. Когда она находит совпадение, она последовательно пытается доказать цель, рассматривая антецеденты совпавшего консеквента в качестве подцелей; если доказана истинность всех подцелей, тем самым доказана и сама цель.

Этот поиск антецедентов для доказательства консеквента очень похож на построение обратной цепочки рассуждений в схеме управления языков, основанных на правилах.

Логические модели являются наиболее строгим, в математическом смысле, способом представления знаний. Но на практике они не получили большого распространения из-за малой наглядности базы знаний. Основная область применения этих моделей – учебные экспертные системы.

2.7. Вопросы для самопроверки и упражнения

1. Что понимается под представлением знаний? Какие требования предъявляются к языку представления знаний? Можно ли утверждать, что представление знаний это кодирование информации на каком-либо формальном языке?

2. Что такое таблица операторов? Можно ли с помощью таблицы операторов представить любое действие робота?

3. Как выполняется поиск цели в системе STRIPS?

4. Можно ли с помощью языка графов описать любые понятия? Что представляют узлы и связи в семантической сети?

5. Какие основные типы отношений должны быть представлены в семантической сети? Приведите пример.

6. Опишите с помощью семантической сети работу университетской библиотеки.

7. Чем отличаются семантические сети и фреймы? Что их объединяет?

8. Какова роль прототипов в теории фреймов? Приведите пример описания прототипа какого-либо объекта или понятия и нескольких фреймов для данного прототипа.

9. Опишите основную идею фреймового подхода к представлению знаний.

10. Какова структура и применение классов в «чистой» объектно-ориентированном парадигме?

11. Приведите определения понятий «наследование», «инкапсуляция» и «полиморфизм».

12. Каково назначение метаклассов в объектно-ориентированных языках?

13. Почему представление знаний на основе продукций получило наибольшее распространение?

14. Как формируется цепочка вывода цели в продукционной системе? Каким образом можно повлиять на ход вывода с помощью метаправил? Приведите пример.

15. На каком формализме основаны логические модели? Почему они не получили большого распространения в коммерческих системах?

3. Логический вывод

На сегодняшний день разработано множество различных методов логического вывода и поиска решений в базе знаний. Определяющим в выборе метода логического вывода является способ представления знаний. Некоторые формы представления, например таблицы решений, предлагают единственный способ поиска решения. Другие формы представления знаний, такие как продукционные правила, не так сильно зависят от деталей реализации машины вывода, и позволяют использовать различные алгоритмы логического вывода с большей или меньшей эффективностью.

В данном разделе будут рассмотрены основные подходы к реализации машины вывода. Поскольку задача вывода по базе знаний фактически является задачей поиска некоторого решения, удовлетворяющего заданным требованиям, то сначала будет рассмотрено понятие задачи и стратегий поиска. Наиболее математически обоснованной, а, следовательно, и более простой для понимания, формой реализации машины вывода является аппарат математической логики. Этот аппарат в основном применяется в учебных задачах, однако принципы математической логики, в большей или меньшей степени, используются во всех алгоритмах логического вывода. Поэтому, наиболее подробно будут рассмотрены приемы логического программирования. Затем последуют описания подходов, применявшихся в некоторых промышленных экспертных системах.

3.1. Понятие задачи поиска в пространстве состояний

Предположим, некоторая задача описана, формализована и для нее существует БЗ правил, позволяющих найти решение этой задачи для некоторых начальных фактов. При реализации машины вывода, позволяющей осуществлять поиск решений, необходимо ответить не следующие вопросы:

- Гарантировано ли нахождение решения в процессе поиска?
- Является ли поиск конечным, или в нем возможны зацикливания?
- Если решение найдено, является ли оно оптимальным?
- Как процесс поиска зависит от времени выполнения и используемой памяти?
- Как можно наиболее эффективно упростить поиск?

- Как наиболее эффективно использовать язык представления знаний?

Для ответов на эти вопросы разработана теория *поиска в пространстве состояний* (state space search) [7]. Представив задачу в виде графа пространства состояний, можно использовать теорию графов для анализа структуры и сложности, как самой задачи, так и способов ее решения.

Рассмотрим в качестве примера простейшую задачу поиска пути на карте из населенного пункта ***A*** в населенный пункт ***G***. При первом взгляде на карту (Рисунок 3-1. Карта) задача кажется элементарной. Однако первая же попытка формализовать подобную задачу стразу же указывает на те потенциальные трудности, которые могут возникнуть при построении алгоритма поиска.

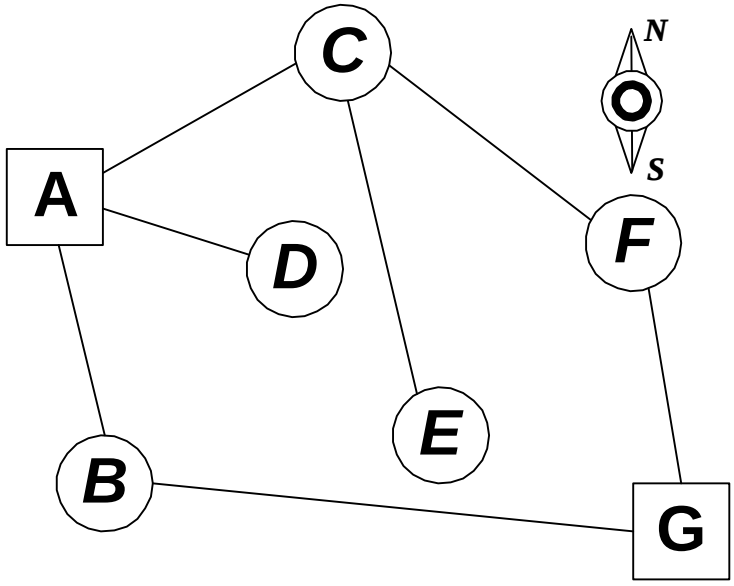


Рисунок 3-1. Карта

Во-первых, с точки зрения алгоритма поиска карта, очевидно, должна выглядеть несколько иначе, нежели чем привычное для нас представление. По меньшей мере, следует учитывать, что никакой алгоритм не может «обозревать» всю карту целиком. А только лишь небольшой ее фрагмент, в простейшем случае только один город. В такой ситуации поиск пути на карте будет напоминать попытку рассмотреть карту в темной комнате с фонариком, освещающим лишь небольшое пятнышко на карте.

Во-вторых, следует учесть, как именно алгоритм будет «воспринимать» карту и «запоминать» свой маршрут. То есть необходимо ввести понятия *состояния среды*, в которой будет работать алгоритм. В процессе поиска решения основная задача разбивается на частные подзадачи. Каждое состояние среды как раз и является решением какой-

либо частной подзадачи, а совокупность всех возможных шагов решения задачи – *пространством состояний*.

Таким образом, поиск решения будет основываться на формализованном, удобном для работы алгоритма представлении задачи, и описании последовательности состояний, ведущих достижению поставленной цели. Процесс поиска задачи также носит название *вывод цели*. Посредством своих действий, алгоритм переводит из оного состояния в другое.

Для решения задачи построим дерево поиска (разновидность графа), или «компьютерную карту», показывающее все возможные варианты маршрутов (Рисунок 3-2. Дерево поиска). При этом важно исключить из рассмотрения маршруты, содержащие замкнутые циклы.

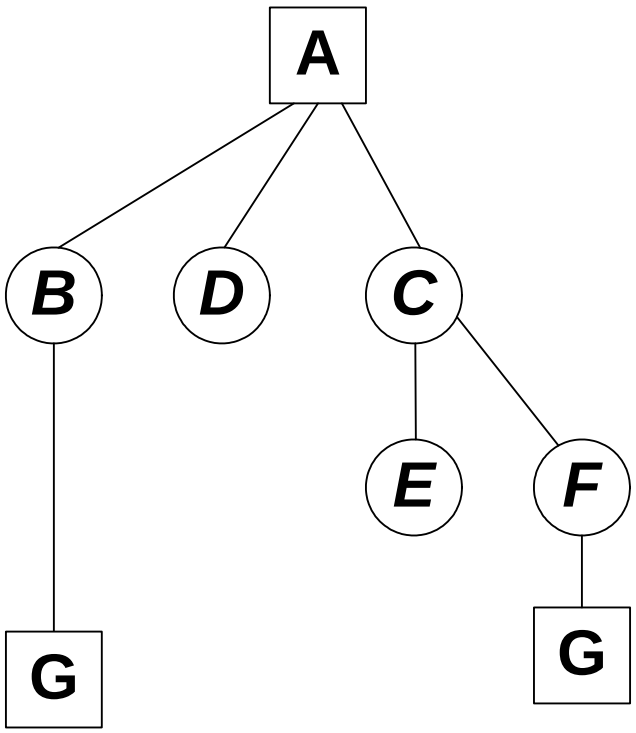


Рисунок 3-2. Дерево поиска

Данное дерево показывает все варианты выбора при движении из ***A*** в ***G***. Начало поиска находится в вершине дерева. Все дороги из ***A*** ведут либо в тупик, либо в ***G***. Следовательно, задача состоит в том, чтобы найти подходящую ветку на графе. Решение этой задачи сводится к последовательному исследованию (перебору) каждой ветви дерева. Для определения последующего шага алгоритма, могут применяться две разновидности стратегий: *поиск в ширину* и *поиск в глубину*.

Стратегия поиска в глубину (depth-first) исследует каждый вариант выбора сверху вниз, пока не зайдет в тупик, или не отыщет решение. Если алгоритм зашел в тупик, то программа возвращается к вершине, где

имеются неизученные ветви и переходит к рассмотрению следующего варианта.

Стратегия поиска в ширину (breadth-first) сначала исследует все варианты, имеющие длину в один маршрут. Затем – имеющие длину в два маршрута и т.д.

Обе стратегии поиска гарантируют рассмотрение всех возможных вариантов. А выбор стратегии определяется спецификой задачи. Так, поиск в глубину подходит для решения проблемы, где все пути поиска имеют приблизительно одинаковую длину. Поиск в ширину пригоден для задач, где ветви дерева поиска имеют сильно различающуюся длину, и нет четких указаний на то, какой путь быстрее приводит к цели. Очевидно, в нашем примере стратегия поиска в глубину даст решение на четвертом шаге, а поиск в ширину – на третьем.

В реальных системах, как правило, не применяется только одна из стратегий поиска. Алгоритмы комбинируют обе стратегии, в зависимости от свойств предметной области и среды поиска. Хорошим примером комбинированного алгоритма поиска является использование предельного значения глубины поиска. *Предельная глубина* позволяет ограничить поиск только заданным числом уровней. Это обеспечивает некоторое подобие «развертки» области поиска в ширину при поиске в глубину.

Как уже было отмечено выше, граф пространства состояний описывает возможные ходы решения задачи, где каждый узел представляет определенно состояние этого решения. В качестве формального языка для описания узлов графа можно использовать исчисление предикатов. Более того, для описания дуг графа или связей между состояниями, можно использовать правила вывода. Тогда методами поиска могут быть решены некоторые проблемы исчисления предикатов. Например, таким образом можно решить, является ли выражение логическим следствием данного набора утверждений. Конкретным воплощением этой идеи является метод резолюций Робинсона, о котором речь пойдет ниже.

Поиск в пространстве состояний можно вести в двух направлениях: от данных задачи к цели и в обратном направлении от цели к исходным данным. Коротко эти способы описывались в параграфе «Продукционные правила» главы «Представление знаний».

При *поиске на основе данных* (data-driven search), процесс решения задачи начинается с исходных фактов. Затем, применяя допустимые правила изменения состояний, осуществляется переход к новым фактам. И так до тех пор, пока цель не будет достигнута. Это процесс также называют *прямой цепочкой вывода* (forward chaining).

При *поиске от цели* (goal-directed strategy) выбирается одна из допустимых целей, и рассматриваются пути, ведущие к достижению этой цели. Таким образом, определяется промежуточный список состояний,

позволяющих найти решение. Процесс повторяется, но теперь уже отталкиваясь от найденного списка, пока не будут достигнуты исходные данных задачи. Такой способ поиска называют также *обратной цепочкой вывода* (backward chaining).

Выбор направления поиска зависит от конкретной задачи, структуры исходных данных и целей, способов реализации алгоритмов поиска. Но можно указать несколько общих соображений, которыми следует руководствоваться при построении алгоритма работы системы логического вывода.

Процесс поиска от цели к данным рекомендован в следующих случаях:

1. Цель поиска явно присутствует в постановке задачи или может быть легко сформулирована. Многие диагностические системы рассматривают возможные диагнозы, систематически подтверждая или опровергая некоторые из них способом поиска от цели.
2. Имеется слишком большое число правил, которые на основе исходных фактов продуцируют возрастающее число заключений или целей. Своевременный отбор целей позволяет отсеять множество тупиковых ветвей, что сокращает пространство поиска.
3. Исходные данные не приводятся в задаче, но подразумевается, что они должны быть известны или могут быть легко получены. Системы медицинской диагностики предлагают множество симптомов или результатов лабораторных анализов, подтверждающих или опровергающих диагноз.

Таким образом, при поиске от цели подходящие правила или шаги решения, в основном, применяются для исключения неперспективных ветвей поиска.

Поиск на основе данных применим к решению задач в следующих случаях:

1. Все или большинство данных заданы в пространстве задачи. Например, задача интерпретации состоит в выборе этих данных и в представлении их для использования в системах интерпретации более высокого уровня.
2. Существует большое количество потенциальных целей, но всего лишь несколько способов представления и применения исходных фактов. Примером является экспертная система DENDRAL, предназначенная для исследования молекулярных структур органических соединений на основе формул, данных масс-спектрографа и знаний из химии. Для любого органического соединения существует чрезвычайно большое число возможных структур. Однако данные масс-спектрографа

позволяют программе оставить лишь небольшое число таких комбинаций.

3. Сформировать цель или гипотезы очень трудно в силу избыточности исходных данных или большого числа конкурирующих гипотез.

Таким образом, при поиске на основе данных исходные факты и ограничения используются для формирования возможных путей решения задачи.

Наконец, следует отметить, что, как правило, в реальных системах используются комбинации обоих способов поиска, наряду с комбинированием стратегий в глубину и в ширину.

3.2. Логика высказываний

Осуществить постановку задачи формально – это значит, имея некий формальный язык, выразить на нём все знания о среде, необходимые для решения задачи. *Формальный язык* определяется через его синтаксис и семантику. *Синтаксис* языка определяет допустимое в языке предложение, состоящее из цепочек символов, принадлежащих определённому множеству, называемому *алфавитом*. Синтаксис языка позволяет отличать предложение, принадлежащее языку от предложений, ему не принадлежащих. *Семантика* языка определяет смысл этих предложений, сопоставляя символы языка с объектами реального мира, а предложение – отношение между объектами. Семантика логики высказываний позволяет разделить всё множество допустимых предложений на истинные и ложные. Истинные – это те предложения, которые соответствуют имеющимся фактам и отношениям, иначе – предложения ложные. Решить задачу формально – это значит, имея множество правил и стратегий их использования, осуществлять вывод синтаксически правильных истинных предложений из других истинных или предполагаемых истинными.

Синтаксис логики высказываний прост и имеет семантические и синтаксические аналоги в естественных языках.

Алфавит языка логики высказываний:

- Логические константы *ИСТИНА* и *ЛОЖЬ*
- Логические переменные, обозначаемые строчными буквами латинского алфавита,
- Логические связки $\wedge$ (И), $\vee$ (ИЛИ), $\neg$ (НЕ), $\equiv$ (ЭКВИВАЛЕНТНО), $\rightarrow$ (СЛЕДУЕТ)
- Круглые скобки.

Значениями логических переменных являются логические константы.

Предложения языка логики высказываний, называемые также формулами или высказываниями, составляют в соответствии со следующими правилами:

1. логические константы и переменные являются простыми предложениями;
2. сложные предложения формируются из простых с помощью логических связок;
3. предложение, заключённое в скобки, также является предложением логики высказываний;
4. из предложений с помощью связок и скобок можно образовать новое предложение;
5. связки имеют следующий приоритет: $\neg \wedge \vee \rightarrow \equiv$.

Предложения, построенные по этим правилам, называют *правильно построенными формулами* или сокращённо *формулами*.

Семантика логики высказываний определяется через *интерпретацию* ее формул, т.е. устанавливает соответствие между логическими переменными и изменяющимися свойствами объектов и между значениями переменных (константами) и значениями свойств. Отношения между объектами определяет взаимосвязь переменных в формуле. Это позволяет по значению формул после подстановки вместо переменных конкретных значений судить о наличии или отсутствии у среды тех или иных свойств или отношений. Подстановка в формулу констант вместо переменных называется *конкретизацией*.

Замечательным свойством логики высказываний является то, что ее семантика близка к соответствующим высказываниям на естественном языке. Так, например семантика формул содержащих связки $\neg$ и $\wedge$ практически совпадает со смыслом фраз содержащих слова «не» и «и». Однако имеются и некоторые различия. Так формула $x \vee y$ несколько шире, чем русское «x или y». Выражение «x или y» по смыслу ближе к формуле $x \wedge \neg y \vee \neg x \wedge$ у. Еще больше различий между семантикой формулы $x \rightarrow y$ в логике высказываний и выражению «из x следует y». В русском языке это выражение истинно, если истинны x и y, т.е. предложение русского языка по смыслу совпадает с формулой $x \wedge y$. Логическое следствие истинно также, если x и y ложны или x ложна, а y истинна. Логическую формулу $x \rightarrow y$ следует интерпретировать на естественном языке так: «Если x истинна, то y тоже истинна, а остальное неизвестно».

Семантика логических формул определяется с помощью *таблиц истинности*. В левой части таблицы перечисляются все значения аргументов формулы, а в правой ее значения соответствующие наборам значений аргументов. Таблицу истинности можно построить для любой правильно построенной формулы. Семантика логических связок представлена в следующей таблице:

Таблица 3-1. Семантика основных логических связок.

x	y	$\neg x$	$x \wedge y$	$x \vee y$	$x \rightarrow y$	$x \equiv y$
Л	*Л*	*И*	*Л*	*Л*	*И*	*И*
Л	*И*	*И*	*Л*	*И*	*И*	*Л*
И	*Л*	*Л*	*Л*	*И*	*Л*	*Л*
И	*И*	*Л*	*И*	*И*	*И*	*И*

Присвоение логическим выражениям значения истина называется *интерпретацией*. Если для логической формулы существует интерпретация, то по таблице истинности можно определить, какие отношения между свойствами объектов, обозначенных переменными, имеют место (формула истинна) и не имеют место (формула ложна).

Формулы, истинные на всех наборах своих аргументов, называют *общезначимыми*. Этот факт обычно записывается: $\models\alpha$ где α – логическая формула. Проверку формулы на общезначимость можно определить с помощью таблицы истинности: если формула истинна на всех возможных аргументов, то эта формула общезначима.

Например, рассмотрим формулу $x \wedge (\neg x \vee y) \rightarrow y$ и её таблицу истинности, представленную ниже в таблице:

Таблица 3-2. Таблица истинности для общезначимо формулы.

x	y	$\neg x$	$\neg x \vee y$	$x \wedge (\neg x \vee y)$	$x \wedge (\neg x \vee y) \rightarrow y$
Л	*Л*	*И*	*И*	*Л*	*И*
Л	*И*	*И*	*И*	*Л*	*И*
И	*Л*	*Л*	*Л*	*Л*	*И*
И	*И*	*Л*	*И*	*И*	*И*

Формулы вида $\alpha_1 \rightarrow \alpha_2$ называются *импликативными*, где α_1 – *посылка*, а α_2 *заключение*. Соответственно связку $\rightarrow$ называют *импликацией*.

В логике высказываний известно много общезначимых формул, называемых *законами логики высказываний*. Наиболее известны следующие:

Таблица 3-3. Основные законы логики высказываний.

Коммутативный закон	$\alpha \wedge \beta \equiv \beta \wedge \alpha$ $\alpha \vee \beta \equiv \beta \vee \alpha$
Дистрибутивный закон	$\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$ $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Ассоциативный закон	$\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$ $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
Закон Де Моргана	$\neg (\alpha \wedge \beta) \equiv (\neg\beta) \wedge (\neg\alpha)$ $\neg (\alpha \vee \beta) \equiv (\neg\beta) \vee (\neg\alpha)$
Закон двойного отрицания	$\neg (\neg \alpha) \equiv\alpha$

В этих законах α обозначает любую правильно построенную формулу логики высказываний.

Кроме общезначимых существуют выполнимые и невыполнимые формулы. Формула называется *выполнимой*, если существуют наборы значений ее аргументов, на которых она принимает истинное значение. Если на всех значениях аргументов формула принимает ложное значение, то она *невыполнима*.

Синтаксис и семантика логики высказываний позволяют представить на этом языке правила решения задачи. Но их не достаточно для осуществления процесса поиска решения. Для осуществления логического вывода используют аппарат, получивший название *логическое исчисление*, который включает в себя:

- Алфавит (совокупность используемых символов);
- Синтаксические правила построения формул в алфавите;
- Аксиомы (общезначимые формулы);
- Правила вывода по аксиомам производных формул или теорем.

Правила вывода вида $\alpha \vdash \gamma$, где α называется *условием*, β – *следствием*, позволяют по истинности α заключить об истинности β. Если в условии или следствии несколько формул, то они записываются через запятую. Если из истинности всех формул, входящих в условие, следует истинность всех формул входящих в следствие, правило называют *состоятельным*. Доказательство состоятельности можно осуществить через построение таблицы истинности, где в строках перечислены все модели условия. Если всем этим условиям соответствуют истинные следствия, то правило состоятельно.

Исчислением высказываний называют исчисление, в котором в качестве алфавита взят алфавит логики высказываний, в качестве синтаксических правил – синтаксические правила логики высказываний, в качестве аксиом – некоторое множество общезначимых формул, а в качестве правил – правила Modus ponens и правило подстановки.

В классическом исчислении высказываний приняты следующие аксиомы:

- $\alpha \rightarrow (\beta \rightarrow \alpha)$,
- $(\alpha \rightarrow \beta) \rightarrow ((\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow (\alpha \rightarrow \gamma))$,
- $(\alpha \wedge \beta) \rightarrow \alpha$,

- $(\alpha \wedge \beta) \rightarrow \beta$,
- $\alpha \rightarrow (\alpha \vee \beta)$,
- $\beta \rightarrow (\alpha \vee \beta)$,
- $\alpha \rightarrow (\beta \rightarrow (\alpha \wedge \beta))$,
- $(\alpha \rightarrow \gamma) \rightarrow ((\beta \rightarrow \gamma) \rightarrow ((\alpha \vee \beta) \rightarrow \gamma))$,
- $(\alpha \rightarrow \beta) \rightarrow ((\alpha \rightarrow \neg\beta) \rightarrow \neg\alpha)$,
- $\neg\neg\alpha \equiv \alpha$.

Классическое исчисление высказываний использует два правила вывода:

- *Modus ponens*. Из истинности условия импликации и истинности самой импликации следует истинность следствия импликации:
  $\alpha, \alpha \rightarrow \beta \vdash \beta$.
- *Правило подстановки*. Из формулы $\alpha(p)$, где p – переменная, выводима формула $\alpha(P)$, где P – формула, получаемая заменой в $\alpha(p)$ каждого вхождения переменной p на формулу P:
  $\alpha(p) \vdash \alpha(P)$.

Правила вывода позволяют в процессе поиска решения заменять одни логические формулы на другие, что является фундаментальной основой для осуществления поиска в пространстве состояний.

3.3. Логика предикатов первого порядка

Недостатком логики высказываний является ее многословность – даже для описания простых задач требуется значительное количество логических переменных и формул. Представление каждого отдельного свойства для каждого конкретного объекта требует отдельной логической переменной, что очень неудобно. Также требуется использовать отдельные переменные для описания всех необходимых комбинаций взаимосвязей между понятиями. С другой стороны, в исчислении высказываний каждый атомарный (элементарный) символ обозначает выражение некоторой произвольной сложности. При этом не существует возможности получить доступ к составным частям этого выражения. Например, фраза «курс рубля имеет тенденцию к росту», обозначенная через логическую переменную r , говорит только о росте курса рубля и не может быть использована в другом контексте. Логика предикатов позволяет решить эти проблемы представления знаний.

Главная идея логики предикатов заключается во взаимнооднозначном сопоставлении каждого уникального объекта с

индивидуальной *объектной константой*, обозначаемой именем объекта, а класс однотипных объектов – с *объектной переменной*, значением которой являются объектные константы. Например, выражение, приведенное выше, можно представить на языке логики предикатов так: *изменение_курса*(*Рубль*, *Растет*). Очевидно, такое представление гораздо более наглядно и гибко.

Предикатом называют высказывательную функцию, определенную на множестве наборов значений объектных переменных. Семантика этой функции определяется предикатным символом, за которым в скобках следуют аргументы (объектные переменные и константы). Эта функция может принимать только два значения: *Истина* или *Ложь*, называемые *истинностными значениями*. Если предикат имеет только один аргумент, то предикатный символ указывает на определенное свойство объекта, а если аргументов несколько – на наличие отношения между объектами, представленными аргументами. Отношения между объектами среды, также как и в логике высказываний, представляются в виде предложений (формул), состоящих из переменных, констант, логических связок, скобок, а также функций, предикатов и кванторов.

Объектная константа или просто *константа* взаимнооднозначно сопоставляется в процессе интерпретации с каким-либо одним объектом среды и обозначается строкой символов, начинающихся с прописной буквы.

Объектные переменные или просто *переменные* обозначаются строкой, начинающейся со строчной буквы. Областью значений каждой переменной является множество констант, в общем случае бесконечное.

Объектные константы и переменные являются *темами*. Как именно выбирать термы для представления знаний – решать разработчику. В приведенном выше примере использовались константы *Рубль* и *Растет*. Введем переменную *x*, определенную на множестве валют и переменную динамики изменения курса *d* = {*Растет*, *Падает*}. Предикатный символ *изменение_курса* ставит во взаимно однозначное соответствие любой валюте свойство плавающего курса. Функция *изменение_курса*(*x*, *d*) задаёт отношение между объектом валюта и динамикой изменения. Если *x* = *Рубль*, *d* = *Растет*, то в соответствии с теми знаниями что у нас есть («курс рубля имеет тенденцию к росту») *изменение_курса*(*Рубль*, *Растет*) = *ИСТИНА*.

С помощью *предикатов* задаются произвольные отношения между объектами. Предикаты начинаются с *предикатного символа* и следующими за ним в скобках упорядоченного набора переменных или констант, соответствующих объектам, которые находятся в поименованном отношении. Так, например, если два человека Маша и Саша являются братом и сестрой, то это отношение может быть выражено с помощью предиката *брат_сестра (Маша, Саша)*. Предикат может

принимать значение *Истинно* или *Ложь*. Если предикат Истинен, то отношение имеет место, иначе – наоборот.

Если некоторый объект в точности соответствует множеству других, то используют *функции*. Например, если объектами являются двоичные цифры и десятичные цифры, то любому двоичному можно однозначно сопоставить десятичное, и выразить это сопоставление в виде функции *преобразование_2_в_10(x, y, z)*, где *x, y, z* – двоичные числа, а значение функции – десятичное. Выражение *преобразование_2_в_10* называют *функциональным символом*. Функция в логике предикатов не предполагает обязательного наличия алгоритма вычисления ее значения по аргументам. Она лишь задает с помощью констант и переменных определенное отношение между объектами, соответствующими ее аргументам, и каким-то одним объектом. Функции также как и переменные или константы являются *темами*.

Выражение *предикатный_символ (терм, терм, …, терм)* называют *атомом*. Атом представляет предикат. Особо выделяется атом, предикатным символом которого является знак равенства, а аргументами два терма. Этот атом можно было бы представить как *равны (терм, терм)* или = *(терм, терм)*, но, как правило, его записывают в обычной инфиксной форме *терм = терм*. Этот атом истинен, когда значения обоих термов совпадают. Атомы без знака отрицания или со знаком отрицания называют *литералами*.

Когда возникает необходимость выразить какие-либо свойства, общие для целого множества объектов, используют *кванторы*. В логике предикатов таких кванторов два: ∀, ∃.

Квантор общности **∀.** Смысл квантора общности совпадает с выражением естественного языка «для всех». То есть, если имеется некоторое знание, применимое для любого объекта определённого типа, то вместо перечисления всех таких объектов можно использовать квантор общности.

Квантор существования **∃.** Если возникает необходимость выразить знание об отдельном объекте из какой-либо совокупности, используют *квантор существования*. Квантор существования произносится на естественном языке как «существует».

Считают, то квантор связывает переменные, которые записываются за знаком квантора в скобках. Поэтому их называют *связанными*. Переменные же, которые ни один квантор не связывает, называют свободными.

Взаимосвязь между кванторами существования и общности можно легко выразить с помощью связки отрицания ¬ и она основана на следующем соображении: если про любой объект из совокупности можно сказать, что он не обладает заданным свойством, то не существует объекта, обладающего этим свойством. Например, очевидно, что «у любой лошади

нет крыльев, значит, не существует лошади, у которой есть крылья». Обозначим любую переменную *x*, а любую формулу, содержащую эту переменную *P(x)*, тогда справедливы следующие законы:

- $\forall (x) \neg P(x) \equiv \neg \exists (x) P(x)$,
- $\neg \forall (x) P(x) \equiv \exists (x) \neg P(x)$,
- $\forall (x) P(x) \equiv \neg \exists (x) \neg P(x)$,
- $\neg \forall (x) \neg P(x) \equiv \exists (x) P(x)$.

Равенство является атомом особого типа *Терм = Терм* или = (*Терм*, *Терм*). Равенство означает, что оба терма в атоме соответствуют одному и тому же объекту. Не следует путать предикат равенства с операцией присваивания. В следующей таблице раскрыт смысл предиката равенства для различных термов, где *X*, *Y* обозначают объектные константы, *x*, *y* – объектные переменные, а *F*(*x*) – функция:

Таблица 3-4. Семантика предиката "равенство".

X = Y	Истинно, если константы именуют один и тот же объект
x = Y	Истинно, если значение переменной равно константе
x = y	Истинно, если значения переменных совпадают
X = F(Y) *X = F(y)*	Истинно, если значение функции совпадает с константой
x = F(Y) *x = F(y)*	Истинно, если значение функции совпадает со значением переменной

Синтаксис логики предикатов позволяет наглядно и просто переходить от естественного языка к языку логики предикатов. Достаточно правильно ввести соответствующие объектные константы и предикатные символы, чтобы иметь возможность корректно описывать состояния и явления предметной области.

Рассмотрим пример. Предположим, что наши знания о птицах выражены в виде следующих предложений:

- Если существо имеет крылья, то это существо – птица.
- Если существо летает и несет яйца, то это существо – птица.

Для представления знаний на языке логики предикатов следует выполнить следующие шаги:

1. Выявить, что во фразе является объектом, который необходимо сопоставить с константой или переменной. Если речь идет о конкретном объекте, то вводится константа, если же упоминается целый класс объектов, то используют переменную.
2. Определить свойства объектов. Сопоставить свойствам предикатные символы.

3. С помощью логических связок сформировать формулы констант, переменных и предикатов, соответствующих объектам и их свойствам.

Таким образом, на языке логики предикатов эти знания могут быть выражены в виде формул:

- *имеет_крылья*(*существо*) → *птица*(*существо*)
- *летает*(*существо*) ∧ *несет_яйца*(*существо*) → *птица*(*существо*)

В данном примере использованы *одноместные предикаты*, имеющие по одному аргументу. Но предикаты могут быть также и *многоместными*, то есть иметь несколько аргументов. В случае многоместных предикатов, как уже было отмечено, предикатный символ может рассматриваться как некоторое общее свойство объектов, соответствующих аргументам, либо как отношение, в котором эти объекты находятся. Вариант первой фразы с применение многоместного предиката может быть, например, следующим:

обладает(*существо*, *крылья*) →
принадлежит_к_классу (*существо*, *птица*).

Выше было рассмотрено исчисление логики высказываний, в частности классическое исчисление. Рассмотрим отличия исчисления предикатов первого порядка от исчислений высказываний.

Аксиомы исчисления высказываний преобразуются в аксиомы исчисления предикатов путем замены $\alpha \Rightarrow \alpha(x)$, то есть логическая переменная α заменяется предикатом $\alpha(x)$. Кроме того, вводятся две новые аксиомы:

- $\forall (x)\ \alpha(x) \to \alpha(y)$,
- $\alpha (y) \to \exists (x)\ \alpha(x)$.

Множество правил вывода включает:

- *Обобщенное правило Modus Ponens*

$$\forall (x_1,\dots x_n, y)\ \alpha(x_1),\dots, \alpha(x_1), \alpha(x_1) \wedge \dots \wedge \alpha(x_1) \to \beta(y) \vdash \beta(y),$$

и *правила введения кванторов*

- $\alpha \to \beta \vdash \alpha \to \forall (x)\ \beta(x)$,
- $\alpha \to \beta \vdash \exists (x)\ \alpha (x) \to \beta$.

Существуют также и неклассические исчисления предикатов первого порядка. Они могут строиться на дополнении множества аксиом специфическими для данной предметной области общезначимыми формулами.

Также могут дополнительно использоваться следующие правила вывода:

- *Исключение квантора общности*: $\forall (x)\ \alpha(x) \vdash \alpha(x \mid A)$,
- *Исключение квантора существования*: $\exists (x)\ \alpha (x) \vdash \alpha(x \mid A)$,
- Введение *квантора существования*: $\alpha (A) \vdash \exists (x)\ \alpha(x \mid A)$.

Здесь $\alpha(x)$ – произвольная формула логики предикатов, имеющая связанную квантором переменную x, $\alpha(x \mid A)$ – формула $\alpha(x)$, в которой все вхождения переменной x заменены на константу A.

Следует особо отметить правило *исключение квантора общности*, которое также называют *правилом универсального инстанцирования*. Другими словами суть этого правила можно сформулировать так: если любую переменную, стоящую под квантором общности в истинном предложении заменить любым соответствующим термом из области определения, то результирующие выражение истинно.

Однако, в отличие от логики высказываний, логический вывод в пространстве предикатов не так очевиден. Чтобы корректно применять правила вывода типа Modus Ponens, система вывода должна уметь определять, когда два выражения являются эквивалентными, или *равносильными*. В исчислении высказываний это тривиально: два выражения равносильны тогда и только тогда, когда они синтаксически идентичны. В исчислении предикатов определение равносильности двух предложений усложняется наличием переменных. Поскольку все термы в логике предикатов являются символьными, то поиск равносильных предложений сводится к процедуре подстановки $\varepsilon(\delta \mid \theta)$, позволяющей заменить терм δ на другой терм из множества θ, подобно тому, как это делается в правиле универсального инстанцирования.

Возможны три вида подстановки $\varepsilon(\delta \mid \theta)$:

1. *Переименование переменной* – вместо переменной δ подставляется переменная из θ.
2. *Конкретизация переменной* – вместо переменной δ подставляется константа из θ.
3. *Замена переменной* – вместо переменной δ подставляется функция из θ.

Обязательным условием подстановки является следующее требование. В пределах области действия подстановки, то есть на множестве всех тех предикатов, к которым она применяется, вместо одной и той же переменной δ подставляется одна и та же переменная, константа или функция из θ для всех ее вхождений δ.

Процесс поиска нужной подстановки называют также процедурой *унификации*. Подстановка называется наиболее общей, если в результате ее применения наименьшее число переменных замещается константами.

Иногда, в правиле Modus Ponens кванторы общности подразумеваются, но не пишутся, т.е. вместо $\forall(x)\ \alpha(x)$ пишут просто $\alpha(x)$, а кванторы существования вообще не употребляются. Если кванторы существования присутствуют в формуле, то от них следует избавиться, например, с помощью правила исключения квантора существования.

Чтобы применять обобщенное правило Modus Ponens, все формулы в постановке задачи должны быть атомами или импликациями, левая часть которых является конъюнкцией атомов, а правая – атомом или пустым символом. Такие формулы получили название *хорновскими предложениями*. Соответственно, для эффективного вывода в пространстве предикатов, необходимо учитывать приведенные выше ограничения и требования к представлению знаний.

Проиллюстрируем работу правила Modus Ponens в логике предикатов на простом примере. Возьмем известный силлогизм «все люди смертны, и Сократ – человек, поэтому Сократ – смертен». В этой фразе присутствуют три основных высказывания – «все люди смертны», «Сократ – человек» и «Сократ – смертен». Очевидно, необходимо ввести константу «Сократ» и предикатные символы «человек» и «смертен». Тогда фраза «все люди смертны» на языке логики предикатов будет выглядеть так:

$\forall (x)$ *человек*$(x) \rightarrow$ *смертен*(x).

А фраза «Сократ – человек» так:

человек(*Сократ*).

Поскольку переменная x связана квантором всеобщности, то ее можно заменить любым значением из области ее определения, например *Сократ*. Следовательно можно сформировать условие правила Modus Ponens:

$\forall (x)$ *человек*$(x) \rightarrow$ *смертен*(x),

человек(*Сократ*),

(x | *Сократ*),

а в результате срабатывания этого правила приходим к выводу *смертен*(*Сократ*).

Несмотря на потенциальные вычислительные возможности правила Modus Ponens на практике оно используется редко. В основном из-за громоздкости вычислений, необходимых для его применения. Основную часть этих вычислений составляет приведение предложений к импликативному виду, необходимому для формирования условия правила. Более мощным, легко реализуемым и требующем меньше вычислений является правило *резолюции*. Это правило легло в основу парадигмы логического программирования, используемого во многих интеллектуальных системах.

3.4. Логическое программирование

Как известно наиболее распространенными являются два типа вывода: прямой и обратный. Правило Modus Ponens позволяет осуществить прямой вывод. Так как в результате работы этого правила

осуществляется последовательный переход от исходных фактов к цели. Обратный вывод работает наоборот: для заданной цели подбираются подтверждающие ее факты. Этот вид логического вывода можно реализовать с помощью правила резолюций. Впрочем, оба типа вывода можно реализовать как с помощью правила Modus Ponens, так и методом резолюций. Кроме того, применение правила Modus Ponens, как уже было отмечено, также связано с вычислительными трудностями, сказывающимися на эффективности вывода.

Для того чтобы сделать вывод более эффективным была предложена стратегия опровержений (refutation). Впервые этот метод предложил в 1965 г. Робинсон, поэтому данная стратегия широко известна как *метод резолюций Робинсона* (resolution refutation).

Идея этой стратегии заключается в опровержении целевой формулы. То есть, если требуется доказать истинность α, то в базу знаний добавляется ее отрицание $\neg\alpha$. Далее на основе обратного вывода, начиная с формулы $\neg\alpha$, предпринимается попытка достичь противоречия, то есть вывод пары одновременно истинных литералов β и $\neg\beta$, что невозможно, так как противоречии здравому смыслу. Такая пара носит название *пустой резольвенты*, так как в результате применения правила резолюций приводит к пустому множеству. Таким образом, делается опровержение истинности $\neg\alpha$, из чего следует истинность $\neg\neg\alpha$ или просто α. Если в результате рассмотрения всех возможных путей вывода противоречия достичь не удается, то считается, что целевая формула является ложной или, иначе говоря, не выводимой.

В основе предложенной Робинсоном стратегии, как это очевидно из названия, лежит метод резолюций. Правило резолюции в логике высказываний имеет следующий вид

$$\alpha \vee \beta, \neg\beta \vee \gamma \vdash \alpha \vee \gamma.$$

В логике предикатов оно выглядит также, но вместо логических переменных α, β, γ в правило входят формулы логики предикатов. Но для применения этого правила, необходимо, чтобы все предложения в базе знаний имели дизъюнктивную форму. Формулы, представленные в виде дизъюнкции литералов называют клаузальными формулами или клаузами (clause). Таким образом, использование правила резолюции требует клаузальной формы представления знаний, что требует подготовки БЗ на начальном этапе. Преимуществом же является то, что в результате применения этого правила форма представления знаний не меняется – выводятся также дизъюнкции. Следовательно, весь вывод может быть осуществлен только на основе правила резолюций. Поэтому нет необходимости осуществлять поиск подходящего правила вывода, а сам вывод превращается в итеративную процедуру выполнения однотипных шагов. Это значительно упрощает реализацию машины логического вывода для метода резолюций, чего нельзя сказать про правило Modus

Ponens. Прежде всего, здесь имеются в виду сложности с подбором фактов для условных частей импликативных формул. В случае если нет точного соответствия условия импликации с какой-либо формулой БЗ, необходимо попытаться сформировать такую формулу. Что ведет к перебору огромного числа комбинаций. И даже, если все импликативные формулы представлены в виде предложений Хорна, то есть условия являются конъюнкциями, и используется обобщенное правило, нет гарантии, что в базе данных будут присутствовать все необходимые конъюнкты. То есть, опять таки придется выводить отдельные конъюнкты из более крупных предложений БЗ.

Различные аспекты применения прямого и обратного вывода можно продемонстрировать на модельной среде кубиков и задачи перестановки эти кубиков из одной комбинации в другую.

Среда состоит из стола и кубиков, расположенных на столе. Каждый из кубиков может лежать либо на столе, либо на другом кубике, образуя, таким образом, столбик. Столбики могут быть любой высоты. Для простоты рассмотрим только столбики высотой в два кубика.

Пусть начальное состояние среды состоит из двух столбиков, так как это изображено на следующем рисунке.

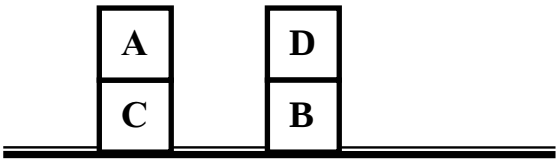


Рисунок 3-3. Начальное состояние для среды кубиков.

Целевое состояние можно представить так:

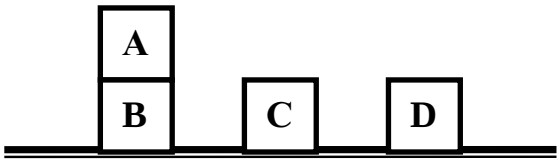


Рисунок 3-4. Целевое состояние среды кубиков.

Введем предикаты *on*(*x*, *y*) и *free*(*x*), обозначающие соответственно расположение одного объекта *x* на другом *y*, и отсутствие какого-либо объекта над *x*. Также введем объектные константы, обозначающие кубики и стол: A, B, C, D, Стол. Тогда начальное состояние среды можно описать следующими атомами:

H1: *on*(A, C),

Н2: *on*(C, Стол),
Н3: *on*(D, B),
Н4: *on*(B, Стол),
Н5: *free*(A),
Н6: *free*(D).

При перемещении кубиков будем руководствоваться следующим правилом: «Если на кубике ничего не лежит (он свободен), то можно его переместить на стол или на другой кубик, на котором ничего не лежит».

В процессе решения задачи необходимо получить последовательность действий, приводящих к целевому состоянию. Для представления действия по перемещению кубиков введем предикат *move*(*x*, *z*).

Рассмотрим прямой вывод целевого состояния с помощью правила Modus Ponens. Для начала введем формулы, представляющие правила возможных действий:

Д1: $on(x, y) \wedge free(x) \rightarrow move(x, \text{Стол})$,

Д2: $on(x, \text{Стол}) \wedge free(x) \wedge free(z) \rightarrow move(x, z)$.

Необходимо также учесть в БЗ изменения среды в результате совершения действий. Формулы для условий перехода состояний среды будут следующими:

П1: $on(x, y) \wedge free(x) \wedge move(x, \text{Стол}) \rightarrow on(x, \text{Стол})$,

П2: $on(x, y) \wedge free(x) \wedge move(x, \text{Стол}) \rightarrow free(y)$,

П3: $on(x, \text{Стол}) \wedge free(x) \wedge free(z) \wedge move(x, z) \rightarrow on(x, z)$.

Наконец, целевая формула может быть представлена следующим предложением в логике предикатов:

$on(\text{A}, \text{B}) \wedge on(\text{B}, \text{Стол}) \wedge on(\text{C}, \text{Стол}) \wedge on(\text{D}, \text{Стол})$.

Начнем вывод с поиска подходящей подстановки, позволяющей использовать имеющиеся начальные факты в условии правила Modus Ponens.

Унификация формулы Д1 приводит к следующим формулам

$on(\text{A}, \text{C}) \wedge free(\text{A}) \rightarrow move(\text{A}, \text{Стол})$,

$on(\text{D}, \text{B}) \wedge free(\text{D}) \rightarrow move(\text{D}, \text{Стол})$.

Шаг 1. На основании этих двух формул, начальных формул расположения кубиков и обобщенного правила Modus Ponens получаем истинные атомы *move*(A, Стол) и *move*(D, Стол). Эти атомы добавляются в базу знаний.

Шаг 2. Далее унифицируются формулы П1 и П2. Аналогично шагу 1, в базу знаний добавляются атомы *on*(A, Стол), *free*(C), *on*(D, Стол), *free*(B).

Шаг 3. Аналогично двум предыдущим шагам на основании формулы Д2 и атомов, полученных на предыдущем шаге, получаем истинность *move*(A, B).

Шаг 4. Унификация формулы П3 приводит к истинность *on*(A, B).

Шаг 5. Наконец, применяя правило введения конъюнкции, получаем целевую формулу.

В ходе прямого вывода, могут быть получены лишние, не используемые в дальнейшем выводе, атомы. Например, *free*(C).

Теперь рассмотрим обратный вывод на основе правила резолюций. Для начала необходимо внести некоторые изменения в БЗ. Учитывая, что

$\alpha \rightarrow \beta \equiv \neg\, \alpha \vee \beta$,

преобразуем импликативные формулы в клаузы:

Д1: $\neg\, on(x, y) \vee \neg\, free(x) \vee move(x,$ Стол$)$,

Д2: $\neg\, on(x,$ Стол$) \vee \neg\, free(x) \vee \neg\, free(z) \vee move(x, z)$.

Формулы для условий перехода состояний среды:

П1: $\neg\, on(x, y) \vee \neg\, free(x) \vee \neg\, move(x,$ Стол$) \vee on(x,$ Стол$)$,

П2: $\neg\, on(x, y) \vee \neg\, free(x) \vee \neg\, move(x,$ Стол$) \vee free(y)$,

П3: $\neg\, on(x,$ Стол$) \vee \neg\, free(x) \vee \neg\, free(z) \vee \neg\, move(x, z) \vee on(x, z)$.

Целевая формула, в соответствии со стратегией метода обратного опровержения должна быть заменена своим отрицанием:

$\neg on$(A, B) $\vee \neg on$(B, Стол) $\vee \neg on$(C, Стол) $\vee \neg on$(D, Стол).

Опровержение этой формулы может осуществляться по очереди, дизъюнкт за дизъюнктом. Рассмотрим обратный вывод на примере опровержения $\neg on$(A, B) для чего добавим в базу знаний литерал ¬on(A, B). Там где необходимо, подразумевается соответствующая унификация. Для удобства изобразим вывод в виде дерева (Рисунок 3-5. Дерево вывода для метода обратного опровержения Робинсона.). Пары, дающие резольвенту, будут подчеркиваться. Пустая резольвента обозначена символом □.

Таким образом, получена пустая резольвента $\neg free$(B), *free*(B), позволяющая опровергнуть предположение $\neg on$(A, B). Аналогично можно опровергнуть $\neg on$(D, Стол), а остальные термы уже имеются в базе знаний. Вывод цели завершен. Прослеживая дерево вывода в обратном порядке снизу вверх, можно сформировать список действий, необходимых для достижения цели. Предикаты, соответствующие этим действиям формировали резольвенты на определенных шагах вывода. В данном случае эта последовательность будет следующей:

move(D, Стол), *move*(A, Стол), *move*(A, B).

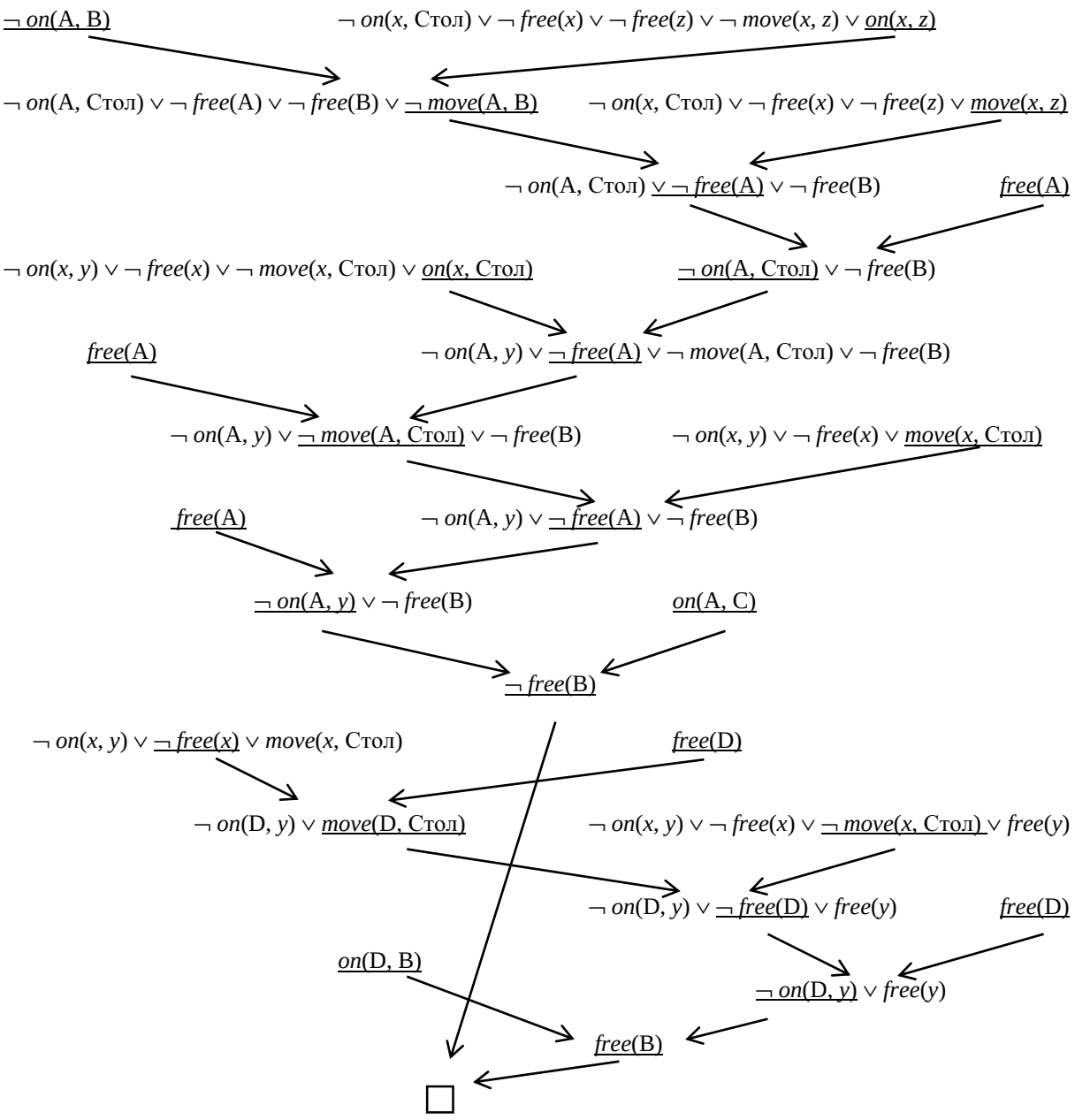


Рисунок 3-5. Дерево вывода для метода обратного опровержения Робинсона.

Подход, описанный выше, был положен в основы парадигмы *логического программирования*. Наиболее известный пример языка логического программирования это PGOLOG (programmation en logique), который был разработан в середине 1970-х во Франции в рамках проекта по пониманию естественного языка.

3.5. Эвристический поиск

Эвристика определяется как «изучение методов и правил открытий и изобретений». Это слово имеет греческий корень, глагол *eurisco* означает «исследовать». Когда Архимед выскочил из ванны, держа золотой венец, он закричал «Эврика!», что значит «Я нашёл!». В пространстве состояний поиска эвристика определяется как набор правил для выбора тех ветвей из пространства состояний, которые с наибольшей вероятностью приведут к приемлемому решению проблемы.

В задачах искусственного интеллекта эвристику используют в двух ситуациях:

1. Проблема может *не иметь точного решения* из-за неопределённости в постановке задачи и/или в исходных данных. Например, в медицинской диагностике определённый набор признаков может иметь несколько причин; поэтому врачи используют эвристику, чтобы поставить наиболее точный диагноз и подобрать соответствующие методы лечения. Система технического зрения – другой пример задачи с неопределённостью. Визуальные сцены часто неоднозначны, вызывают различные (часто не связанные друг с другом) предположения относительно протяжённости и ориентации объектов. Оптический обман – хорошая иллюстрация таких двусмысленностей. Системы технического зрения часто используют эвристику для выбора наиболее вероятной интерпретации из нескольких возможных.
2. Проблема может иметь точное решение, но *стоимость его поиска* может быть *слишком высокой*. Во многих задачах (например, игра в шахматы) рост пространства возможных состояний носит экспоненциальный характер, другими словами, число возможных состояний растёт экспоненциально с увеличением глубины поиска. В этих случаях методы поиска решения «в лоб» типа поиска в глубину или в ширину могут занять слишком много времени. Эвристика позволяет избежать этой сложности и вести поиск по наиболее «перспективному» пути, исключая из рассмотрения неперспективные состояния и их потомки. Эвристические алгоритмы могут (как правило, на это надеются проектировщики) остановить этот комбинаторный рост и найти приемлемое решение.

Эвристика и разработка алгоритмов для эвристического являются основным объектом исследования в проблемах искусственного интеллекта. Невозможно рассмотреть каждый вывод, который может быть сделан в той или иной области математики, или каждый возможный ход на шахматной

доске. Эвристический поиск – часто единственный практический метод решения таких проблем.

Исследование в области экспертных систем подтвердило важность эвристики как основного компонента при решении задач. Эксперт-человек, решая проблему, анализирует доступную информацию и делает вывод. Интуитивные правила (rules of thumb), которые использует эксперт, чтобы эффективно решить ту или иную проблему, в значительной степени эвристичны по природе. Такие эвристики создаются и формализуются разработчиками экспертных систем.

Слепой поиск решения в пространстве состояний возможен только в небольшом пространстве вариантов. Напрашивается совершенно естественный вывод, что необходим некоторый способ направленного поиска. Если такой способ использует при поиске пути на графе в пространстве состояний некоторые знания, специфические для конкретной предметной области, его принято называть *эвристическим поиском*. Лучше всего рассматривать эвристику в качестве некоторого правила влияния, которое, хотя и не гарантирует успеха (как детерминированный алгоритм или процедура принятия решения), в большинстве случаев оказывается весьма полезным.

Простая форма эвристического поиска – это *восхождение на гору* (hill climbing). В процессе поиска в программе использует некоторая оценочная функция, с помощью которой можно грубо оценить, насколько «хорошим» (или «плохим») является текущее состояние. Затем можно применить ту же функцию для выбора очередного шага, переводящего систему в следующее состояние.

Основной алгоритм, реализующий идею восхождения на гору, можно сформулировать следующим образом.

1. Находясь в данной точке пространства состояний, применить правила порождения нового множества возможных решений.
2. Если одно из новых состояний является решением проблемы, прекратить процесс. В противном случае перейти в то состояние, которое характеризуется наивысшим значением оценочной функции. Вернуться к шагу (1).

Но применение этого подхода наталкивается на хорошо известные трудности. Главная из них – как сформулировать оценочную функцию, которая адекватно бы отражала «качество» текущего состояния. Например, для игры в шахматы известно, что иметь больше фигур, чем у соперника, отнюдь не значит иметь лучшую позицию, то есть быть ближе к успеху. Такая простая оценочная функция не учитывает многих особенностей этой игры (а в более широком контексте – особенностей данной предметной области).

Более того, даже если оценочная функция и позволяет адекватно оценить текущую ситуацию, существуют разнообразные ситуации,

которые сами по себе могут быть источником затруднений. Например, в данном состоянии нет очевидного очередного хода, т.е. оказывается, что все возможные ходы одинаково хороши (или плохи). Это не что иное, как выход на «плато» в нашем восхождении, когда ни одни из возможных путей не влечёт за собой подъём. Другой возможный источник затруднений – наличие *локальных максимумов*, из которых возможен только спуск, т.е. «ухудшение» состояния. Например, можно взять вашего ферзя и после этого проиграть партию.

Лучшими свойствами обладает другая форма эвристического поиска, которая получила наименование *сначала наилучший* или *жадный алгоритм* (best-first search). Так же, как и в варианте восхождения на гору, в нашем распоряжении имеется оценочная функция, с помощью которой можно сравнивать состояния в пространстве состояний. Основное же отличие нового метода от ранее рассмотренного состоит в том, что сравниваются не только те состояния, в которые возможен переход из текущего, но и все, до которых «можно достать». Такой алгоритм, естественно, требует гораздо больших вычислительных ресурсов. Идея состоит в том, чтобы принимать во внимание не только ближайшие состояния, то есть локальную обстановку, а рассмотреть как можно больший участок пространства состояний. В случае необходимости, предусматривается возможность возврата на одну из предыдущих позиций, и продолжить поиск другим путём, если ближайшие претенденты не сулят существенного прогресса в достижении цели. Вот эта возможность отказаться от части пройденного пути во имя глобальной цели и позволяет найти более эффективный путь. Конечно, есть и свои издержки: необходимость хранить ранее сделанные оценки состояний и постоянно их обновлять требует значительных вычислительных ресурсов.

В качестве примера рассмотрим игру в крестики нолики. Пространство поиска будут составлять возможные ходы во время игры. Исчерпывающий поиск (полный перебор) в этой игре затруднителен, но возможен. Количество шагов при полном переборе посчитать нетрудно. Изначально имеется девять возможных ходов. На каждый из этих ходов возможны восемь вариантов ходов противника. Далее остается семь различных ходов и так далее. Таким образом, пространство поиска включает $9 \times 8 \times 7 \times 6 \times \ldots = 9! = 362880$ состояний.

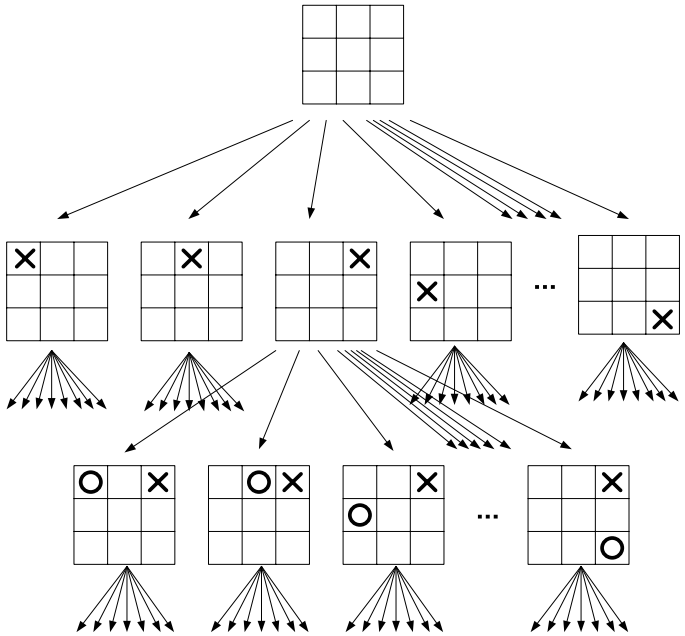


Рисунок 3-6. Все возможные варианты ходов в игре "крестики-нолики".

Попробуем сократить пространство поиска за счет введения эвристик. Первая эвристика может основываться на симметрии задачи. То есть множество различных ходов можно уменьшается за счет симметричной конфигурации игрового поля. В действительности существует только три возможных начальных хода, а не девять, как это кажется на первый взгляд – в угловую клетку, в центр и в центральную клетку на боковой линии. На втором ходе имеется только 12 различных ситуаций, а не 9×7. Таким образом, пространство поиска уже сокращено до $12 \times 7! = 60480$ состояний или в 6 раз.

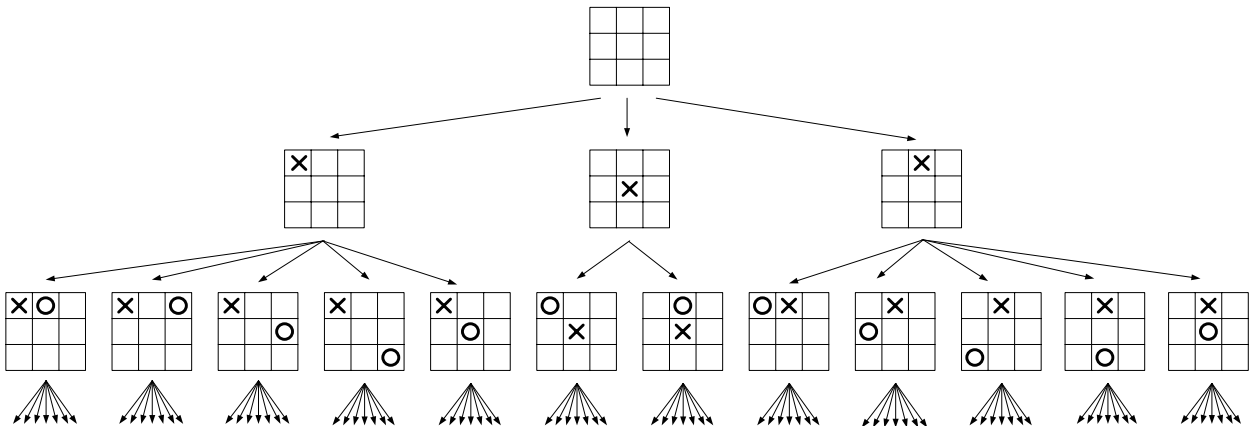


Рисунок 3-7. Первые три уровня пространства состояний в игре "крестики-нолики", редуцированные с учетом симметрии.

Далее можно ввести еще одну эвристику: можно совершать только такие ходы, которые дают наибольшее число выигрышных ситуаций – заполненных линий (Рисунок 3-8. Применение эвристики "максимума выигрышных линий" в игре "крестики-нолики".). Если число потенциальных побед у нескольких комбинаций равно, то берется первый вариант. Заметим, что отбрасываются не только неперспективные комбинации, но и их потомки. Таким образом, полный перебор не требуется.

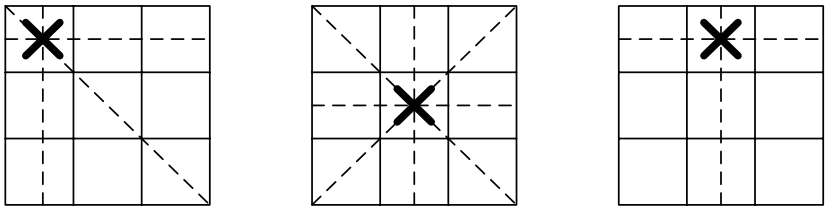


Рисунок 3-8. Применение эвристики "максимума выигрышных линий" в игре "крестики-нолики".

Противник после первого хода может выбрать один из двух возможных вариантов ответных ходов (в силу симметрии рассматриваются только два из восьми). Для любого хода оппонента можно применить эвристику «максимума выигрышных линий», и вычислить эвристическую оценку (Рисунок 3-9. Эвристическая редукция пространства состояний в игре "крестики-нолики".). Следовательно, из всех возможных вариантов будет выбираться наилучший для текущей ситуации.

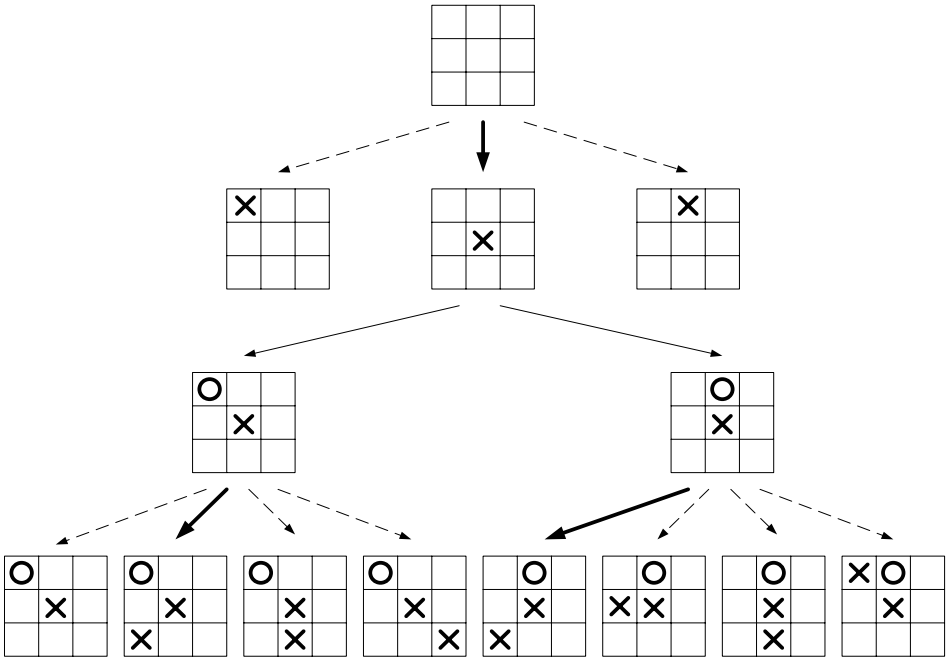


Рисунок 3-9. Эвристическая редукция пространства состояний в игре "крестики-нолики".

Так как не известны ходы противника, затруднительно вычислить количество возможных состояний в пространстве поиска с учетом приведенных эвристик. Но можно вычислить оценку сверху, если допустить, что максимальное количество ходов – 9, и каждый ход имеет по 8 вариантов. В действительности ходов гораздо меньше, так как поле в процессе игры заполняется и половину ходов делает противник. Но даже при грубой оценке сверху получается 8 × 9 = 72 состояния, что на 4 порядка (в 5040 раз) меньше, чем 9!.

3.6. Символьные вычисления и функциональное программирование

Различные способы представления знаний и алгоритмы вывода на этих знаниях оперируют с множеством упорядоченных наборов символов. Символы формируют слова, слова – предложения. Предложения, как правило, однозначно могут быть интерпретированы пользователем экспертных систем, так как их семантика соответствует семантике аналогичных предложений в естественном языке. Логика предикатов, рассмотренная выше, является формализмом, позволяющим осуществлять логические вычисления над символьными структурами, имеющими форму предикатов. Однако, язык PROLOG, основанный на этом формализме, не позволяет выйти за рамки аппарата математической логики. Это

затрудняет использование языка PROLOG для реализации отличных от метода резолюций схем логического вывода и обработки структур данных, не являющихся предикатами. В частности, написанные на PROLOG-е эвристические алгоритмы могут оказаться весьма трудно понимаемыми. А некоторые алгоритмы символьных вычислений эффективно реализовать на этом языке невозможно. Хотя, справедливости ради, стоит отметить, что в силу встроенных в язык средств представления и вывода, тексты логических программ (задач, формализованных посредством предикатов) написанных на PROLOG-е будут значительно нагляднее и короче чем на LISP.

Более чем за 15 лет до появления PROLOG-а, в конце 1950-х, был создан язык LISP (List Processing). Этот язык хорошо зарекомендовал себя как язык, позволяющий осуществлять любые символьные вычисления, что сделало его одним из самых распространенных инструментальных средств для задач искусственного интеллекта. Более того, успех применения этого языка на практике привел к созданию специализированных символьных компьютеров – LISP-машин, реализующих на аппаратном уровне вычислительные операции над символьными структурами.

Основными уникальными особенностями LISP, отличающими его от прочих языков, являются следующие:

- Основной структурой данных является список символов,
- Программы на этом языке также имеют списочную структуру,
- Базовыми операциями являются операции над списками.

В 1960 году выбор списков в качестве базовой структуры казался революционным шагом в программировании. Сегодня практически все языки поддерживают операции со списками, но именно благодаря LISP появилась *парадигма функционального программирования*, основанная на математической теории рекурсивных функций. Входными и выходными данными этих функций являются списочные структуры. Сами функции также представлены в виде списка других функций. Таким образом, на входе программы (списка) может быть другая программа (другой список), изменяющая ход вычислений. Возможность влиять на ход вычислений посредством структуры входных данных и является особенностью, обеспечивающей фундаментальные возможности LISP. Такой способ организации программ также известен под названием *вычисление, управляемое данными*.

Мощь функционального программирования в сочетании с богатым набором высокоуровневых средств построения символьных структур данных, позволяют воплощать с помощью LISP любые формы представления знаний: предикаты, фреймы, сети и объекты, а также алгоритмы их обработки.

Итак, в основе символьных вычислений, а значит и во всех интеллектуальных задачах, лежит понятие *символа*. Неформально можно

определить символ как «нечто, заменяющее другое нечто», например, некоторая последовательность букв и цифр соответствующая объекту реального или вымышленного мира. «Другое нечто» в данном случае является *значением* (designation) символа – то на что ссылается и что представляет символ.

Идея символьных вычислений состоит в том, что мы можем понимать под символами, с которыми выполняются какие-либо действия, все, что угодно. Программа, обрабатывающая эти символы, позволяет связывать одни символы с другими, манипулировать с символами, создавать структуры символов и пр. Эти операции изменяют семантику символов, имитируя тем самым деятельность человека при решении задач.

Синтаксическими элементами языка LISP являются *символьные выражения* (symbolic expression), которые называются *s-выражениями*. В виде s-выражений представляются и данные, и программы. S-выражение может быть либо *атомом* (atom), либо *списком* (list). Ниже приведены несколько примеров атомов:

```
3,1415
x
100
good
*слово_на_русском_языке*
nil
```

Последний атом является специальным атомом в LISP, обозначающим пустой список. Это единственное s-выражение, являющееся одновременно и атомом и списком. Все остальные списки в LISP формируются из атомов или других (вложенных) списков, разделенных пробелами и ограниченных круглыми скобками. Пустой – `nil` – список можно обозначить (). Благодаря этим скобкам LISP получил неформальное название «язык скобок». Ниже следуют еще несколько примеров списков:

```
(1 2 3 4 5)
(tom mary john joyce)
(a (b c) (d (e f)))
```

Глубина вложенности списков может быть произвольной. Список, кажущийся на первый взгляд примитивной структурой, является чрезвычайно гибким инструментом представления. Например, его можно использовать для представления выражений из теории предикатов:

```
(on block-1 table)
(likes bill X)
(and (likes george kate) (likes bill merry))
```

Как уже отмечалось, программы на LISP также являются списками. Например, программы, выполняющие арифметические действия в префиксной форме, могут выглядеть так:

```
(* 7 9)
(- (+ 3 4) 7)
```

В этих примерах символы арифметических операций являются именами соответствующих функций. Интерпретатор LISP-а может вычислить значения этих функций по передаваемым параметрам, следующим за именем в том же списке. Таким образом, вызов функции – это тоже список.

Не всегда требуется сразу же вычислять значение функции или, другими словами, оценивать s-выражение. В LISP имеются две специальные функции `eval` и `quote`, которые позволяют соответственно оценить s-выражение или не делать этого. Например, результатом вычисления функции `(eval (+ 2 3))` будет `5`, а `(quote (+ 2 3))` – `(+ 2 3)`.

Для определения новых функций предназначена функция `defun`. Например, если функцию вычисления квадрата числа можно определить так

```
(defun sqr (x) (* x x))
```

тогда значением выражения `(sqr 5)` будет `25`.

Однако, это не единственный, и не главный способ определения функций. Зачастую необходимо передавать в качестве параметра само определение функции, не определяя ее глобально. Осуществить это можно с помощью *лямбда-исчисления* (lambda-calculus) – математической моделью обеспечивающей четкое разграничение между объектом и его именем. Синтаксис лямбда-выражения представляет собой список следующей структуры

```
(lambda (<функциональные параметры>) <тело функции>).
```

Например, вычисление квадратного корня без определения функции можно осуществить так:

```
(funcall #'(lambda (x) (* x x)) 4).
```

Функция `funcall #'` осуществит вычисление переданного лямбда-выражения для параметра 4. Конечно, через лямбда-выражение можно определить и именованную функцию:

```
(defun sqr (x) (lambda (x) (* x x))).
```

Важнейшим элементом любого императивного языка программирования являются операторы условия. Аналоги таких операторов имеются и в LISP-е, но, естественно, в виде функций и списков. Реализуется ветвление на базе функции `cond`, аргументами которой являются пары условие – действие:

```
(cond (<условие1> <действие1>)
      (<условие2> <действие2>)
                  …
      (<условиеN> <действиеN>))
```

Условия и действия могут быть произвольными s-выражениями, при этом каждая пара заключается в скобки. Функция `cond` не оценивает свои аргументы. Вместо этого она оценивает по порядку переданные ей условия до тех пор, пока не найдет такое, которое возвратит ей список отличный от `nil`. В этом случае оценивается соответствующее действие, и полученный результат возвращается в качестве значения выражения `cond`. если же все условия при оценивании вернут `nil`, то и `cond` вернет `nil`. В качестве примера использования функции `cond` рассмотрим определение модуля числа:

```
(defun abs (x)
    (cond ((< x 0) (- x))
          ((>= x) x)))
```

Часто возникает рассмотреть одно или несколько условий, для которых требуются специфические вычисления, а для остальных случаев следует выполнять какое-либо общее действие. Эту ситуацию можно реализовать с помощью специального атома T, который в языке LISP соответствует логической истине:

```
(defun abs (x)
    (cond ((< x 0) (- x))
          (T x)))
```

Возможности языка LISP позволяют выполнить любые задачи на символьном уровне представления. Однако следует четко понимать, что проектирование экспертных систем должно вестись на уровне знаний, с учетом специфики и ограничений предметной области. В языке LISP нет ничего такого, что могло бы подсказать, как лучше организовать знания. Списочные структуры это удобный инструмент для представления множеств символов. Функциональное программирование – это мощный аппарат манипулирования этими списками. Однако по отдельности эти средства есть практически в любом современном языке программирования. Но совокупность всех качеств LISP сделала его удобным и эффективным инструментом для создания оболочек экспертных систем, так как гибкость языка позволяет легко и быстро создавать на его основе произвольные языки описания знаний и системы логического вывода.

3.7. Системы с доской объявлений

Большинство экспертных систем используют продукционные правила (production rule) в качестве языка представления знаний. Вывод в таких системах может быть организован с помощью цикла *распознавание-действие*, в процессе которого текущее состояние решения задачи

представляется в памяти в виде множества так называемых образцов. Эти образцы сопоставляются с условиями продукционных правил, что порождает набор возможных действий. Этот набор получил название *конфликтное множество*. А продукции, содержащиеся в конфликтном множестве, называются *допустимыми*. Далее по определенному алгоритму (возможно, с использованием сложных эвристик) выбирается одно из правил и выполняется его действие. Этот процедура названа *разрешением конфликтов* (conflict resolution), а выполнение правила – *активацией* или *возбуждением правила*. Активированное правило в свою очередь изменяет содержимое рабочей памяти. Системы, основанные на этом принципе, получили названия *продукционные системы*.

В процессе формирования БЗ можно столкнуться с трудностями при обобщении различных подходов к решению задачи. Если в процессе решения задействованы несколько альтернативных ходов рассуждений, возможно основанных на различных наборах исходных данных, то единое множество правил может оказаться сильно противоречивым и не приводить к решению вообще. В таком случае объединение групп правил, относящихся к различным подходам нецелесообразно.

Методология *доски объявлений* или *классной доски* (blackboard) позволяет расширить возможности продукционных систем за счет организации рабочей памяти в виде отдельных модулей, соответствующих определенным подмножествам правил. Доска объявлений интегрирует эти модули и координирует действия нескольких параллельных решателей задач в рамках единой глобальной структуры.

Этот процесс напоминает обсуждение проблемы группой различных экспертов, сидящих возле классной доски, являющихся специалистами в какой-то определенной области и имеющих отношение к решению проблемы. Формулировка проблемы и исходные данные записаны на доске. Эксперты по очереди анализируют то, что написано на доске. Если кто-либо знает, что можно привнести в решение проблемы, то он выполняет соответствующие вычисления и записывает результаты на той же доске. Эти новые результаты позволяют другим экспертам внести свой вклад в решение проблемы. Наконец, процесс прекращается, когда проблема будет решена или когда будут исчерпаны все возможности (правила) для поиска решения задачи.

Такая методика совместного решения проблем будет эффективна, если выполняются следующие требования:

- Различные модули правил должны иметь сходные синтаксис обозначений, имена объектов и пр., чтобы обеспечить передачу знаний между модулями. Хотя при записи результатов на доске могут использоваться и разные схемы обозначений.
- Должен существовать протокол определения очередности выполнения правил, который вступает в силу в ситуации, когда

сразу несколько альтернативных модулей готовы изменить содержимое доски объявлений.

С точки зрения организации процесса вычислений, в системе можно выделить следующие структурные компоненты (Рисунок 3-10. Архитектура доски объявлений.). Знания о предметной области разделены между независимыми *источниками знаний* (KS_i – knowledge sources), которые работают под управлением *планировщика* (scheduler). Решение формируется в некоторой глобально доступной структуре – *доске объявлений* (blackboard).

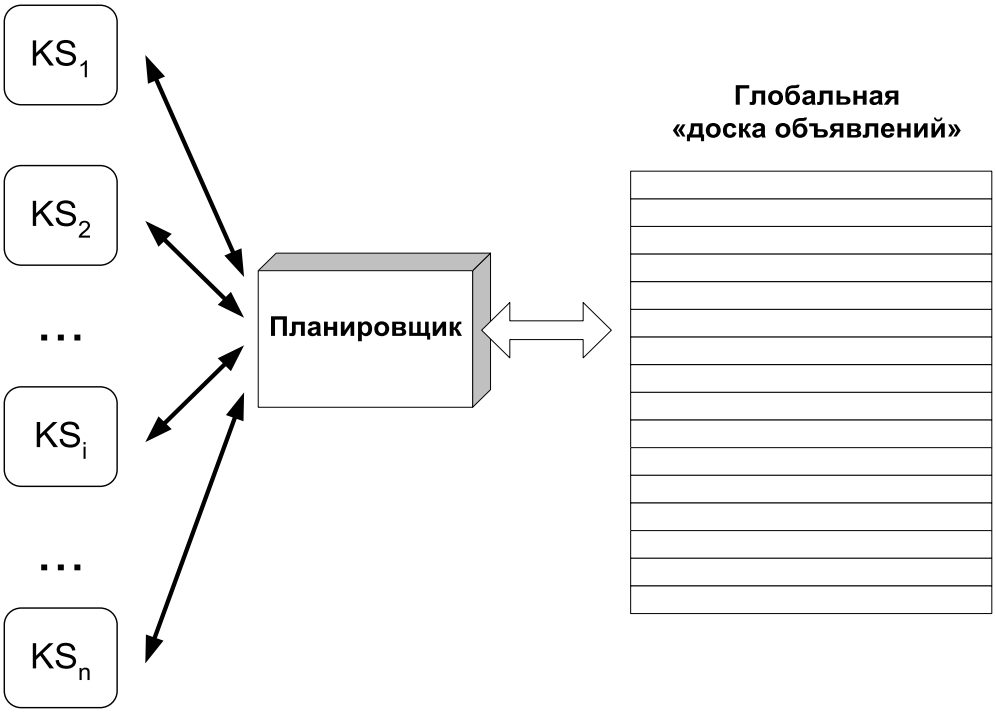


Рисунок 3-10. Архитектура доски объявлений.

Функции доски объявлений во многом сходны с функциями рабочей памяти в продукционных системах, но ее организационная структура значительно сложнее. Как правило, доска объявлений разделяется на несколько *уровней* описания, причем каждый уровень соответствует определенной степени детализации. Данные в пределах отдельных уровней доски объявлений представляют иерархии объектов или графы. В самых современных системах может быть даже несколько досок объявлений.

Источники знаний формируют объекты на доске объявлений, но это выполняется посредством планировщика. Активированные правила помещаются в специальный *список выбора* (agenda), откуда их извлекает планировщик. Источники знаний общаются между собой только через доску объявлений и не могут непосредственно передавать данные друг другу или запускать выполнение каких-либо процедур. Здесь есть определенная аналогия с организацией работы продукционных систем, в

которых правила также не могут непосредственно активизировать друг друга: взаимодействие проходит через рабочую память.

Доска объявлений работает по асинхронному принципу – каждый источник знаний KS_i начинает свою работу, когда находит соответствующие входные данные на доске объявлений. Закончив работу, он возвращает на доску полученные результаты и ожидает новых входных данных.

Архитектура доски объявлений показала высокую эффективность при решении задач, так как она значительно упрощает разработку базы знаний. Однако модульная организация памяти требует больших ресурсов, вследствие чего системы, основанные на этом принципе, работают достаточно медленно. Тем не менее, этот подход считается весьма перспективным и остается областью активных исследований в искусственном интеллекте.

3.8. Вопросы для самопроверки и упражнения

1. Какие основные вопросы задачи поиска необходимо рассмотреть для создания эффективной машины вывода?

2. Назовите и опишите работу основных стратегий поиска в пространстве состояний?

3. Приведите пример комбинированного алгоритма поиска. Почему комбинированные алгоритмы являются более предпочтительными при решении реальных задач?

4. Чем отличается прямая цепочка вывода от обратной? В каких случаях следует применять первую, а в каких – вторую?

5. Приведите описание синтаксиса и семантики логики предикатов. Чем логика предикатов отличается от логики высказываний?

6. Выразите с помощью предикатов следующие утверждения: «Каждый студент использует какой-нибудь компьютер, и, по крайней мере, один компьютер используется каждым студентом», «Каждый год хотя бы один студент проваливает все экзамены, и каждый год хотя бы одна студентка сдает все экзамены». Постарайтесь использовать минимальное количество предикатов. Особое внимание уделите правильному введению кванторов.

7. В чем заключается суть идеи стратегии опровержения в методе резолюций Робинсона?

8. Каким образом можно преобразовать импликативные фразы Хорна в клаузальные формулы?

9. Попытайтесь унифицировать и привести наиболее общую подстановку для следующих пар предикатов или объясните, почему они не могут быть унифицированы:
   - $p(X, Y)$ и $p(a, Z)$
   - $p(X, X)$ и $p(a, b)$
   - $p(X, Y)$ и $p(a, a)$
   - $q(X)$ и $\neg\, q(a)$

10. Приведите в виде дерева вывод целевого состояния on(A, Стол) $\wedge$ on(B, Стол) $\wedge$ on(C, Стол) $\wedge$ on(D, Стол) для примера «среды кубиков».

11. В каких случаях применяются эвристики в задачах искусственного интеллекта?

12. Какие алгоритмы поиска называются эвристическими? Назовите и кратко опишите некоторые.

13. Какими особенностями обладают программы на LISP? Какая структура и как используется в качестве основы этого языка?

14. Для чего необходимо такое лямбда-исчисление? Приведите пример.

15. Опишите архитектуру доски объявлений. Каким образом можно осуществить параллельные вычисления, используя данный подход?

4. Представление нечетких знаний

При решении реальных задач часто возникают ситуации неопределенности, которые можно разделить на две категории: отсутствие достаточно полного и достоверного знания о предметной области и отсутствие возможности получить исчерпывающую информацию о конкретном состоянии среды, объекте, ситуации и т.п.

В первом случае речь может идти о плохо изученных явлениях, противоречивых теориях или нечетко сформулированных концепциях. Так, например, применение в терапии новых препаратов часто дает совершенно неожиданный результат, который невозможно предсказать. Возможна также и противоположная ситуация: предметная область хорошо изучена, но эксперты предпочитают прибегать к неформальным, но более эффективным, эвристическим приемам, вместо использования рутинных точных методов. Например, при поиске неисправности в электрической схеме, как правило, заменяют несправный блок целиком, вместо поиска сгоревших узлов.

Существуют и более прозаичные источники неопределенности знаний. Типичный пример – это ненадежные или неточные данные. Любая информация, поступающая с датчиков, обладает некоторой погрешностью. Социологические опросы также подразумевают некоторый процент ошибок, обусловленных «человеческим фактором». Информация может быть зашумлена так, что полностью полезный сигнал невозможно выделить из общего потока данных. Возможно, также, что приходиться пользоваться информацией, полученной ранее, и которую невозможно не проверить, не дополнить не получить повторно.

Для преодоления проблемы неопределенности знаний в области искусственного интеллекта были разработаны различные методы, применяемые при построении экспертных систем. Наиболее неформальный подход – это использование коэффициентов уверенности, выражающих степень достоверности знания. Альтернативный способ заключается в использовании теории вероятности. Однако не ясно, как с помощью вероятности представить такие понятия как «часто», «иногда», «старый», «высокий» и т.п. Кроме того, теория вероятности подразумевает значительное количество вычислений, для обновления вероятностных оценок. Широкое распространение получили также аппараты нечеткой логики теории и функций доверия. Однако в последние годы внимание исследователей все больше привлекает теория вероятностей, с ее развитым и строго формализованным аппаратом.

4.1. Коэффициенты уверенности

Использование коэффициентов или степеней уверенности можно продемонстрировать на примере системы MYCIN. Типичное правило этой системы выглядит следующим образом:

> Если ***условие*$_1$** и…и ***условие*$_m$**,
> то прийти со степенью уверенности ***x*** к ***заключение*$_1$** и … и к ***заключение*$_n$**.

Степень уверенности характеризует меру правдоподобия того или иного заключения, изначально задаваемую экспертом. Возможность применения данного правила определяется удовлетворением условий с некоторым уровнем истинности. То есть истинность утверждений, содержащихся в предпосылках правила, может также иметь нечеткий характер, например, в силу предыдущих шагов вывода или из-за неточности источника фактов. После применения подобных правил к имеющимся фактам формируется более общее правило, включающее также оценку истинности соблюдения условий:

> Если ***условие*$_1$** удовлетворяется с истинностью ***x*$_1$** и … и ***условие*$_m$** удовлетворяется с истинностью $\mathbf{x_m}$,
> то прийти к ***заключению*$_1$** со степенью уверенности ***y*$_1$** и … и к ***заключению*$_n$** со степенью уверенности ***y*$_n$**.

В системе MYCIN коэффициенты уверенности (степени уверенности, истинность условий) – *CF* (certainty factor) – могут принимать значения в диапазоне от -1 до 1. В общем случае можно использовать любой диапазон значений. Положительные значения выражают уверенность эксперта в фактах, заключениях и пр. Отрицательные значения, напротив, выражают ошибочность утверждений.

Окончательный коэффициент уверенности всего правила вычисляется как произведение: *CF* (заключение) = *CF* (предпосылки) · *CF* (правила).

Если в БЗ найдется несколько правил для данной предпосылки, то в системе MYCIN заключения этих правил объединяются. Пусть X и Y – коэффициенты уверенности одинаковых заключений, полученные при применении разных правил, тогда

$$CF(X,Y)=\begin{cases} X+Y-XY, & при \quad X,Y>0 \\ X+Y+XY, & при \quad X,Y<0 \\ (X+Y)/(1-\min(|X|,|Y|)), & при \quad (X>0 \; и \; Y<0) \quad или \quad (X>0 \; и \; Y<0) \end{cases}$$

Очевидно, что формула обладает свойством коммутативности, и порядок следования гипотез не имеет значения.

Вычисление коэффициентов уверенности имеет модульный характер, то есть при вычислении достаточно той информации, что уже содержится в правиле. При этом не имеет значения, как были получены

коэффициенты уверенности, характеризующие исходные данные. Эта особенность часто используется при построении ЭС – предполагается, что для всех правил, имеющих дело с определенным параметром, предпосылки этих правил логически независимы. Если же имеет место зависимость между условиями правил, то следует перейти к более общему правилу. То есть вместо нескольких правил вида

Если ***условие*$_1$**,
то прийти со степенью уверенности ***x*$_1$** к ***заключение***.
…
Если ***условие*$_m$**,
то прийти со степенью уверенности ***x*$_n$** к ***заключение.***

Следует использовать одно правило вида:

Если ***условие*$_1$** и … и ***условие*$_m$**,
то прийти со степенью уверенности ***x*** к ***заключение***.

В основе этой идеи лежит одно из следствий теории вероятностей, гласящее, что $P(H \mid E_1, E_2)$ не может быть простой функцией от $P(H \mid E_1)$ и $P(H \mid E_2)$.

4.2. Условная вероятность и правило Байеса

Для представления неопределенности знаний можно весьма эффективно использовать положения теории вероятностей. Подобные представления базируются на понятии условной вероятности. Как следует из определения, условная вероятность события d при данном s – это вероятность того, что событие d наступит при условии, что наступило событие s. Например, условной вероятностью является вероятность того, что у пациента действительно имеется заболевание d, если у него обнаружен только симптом s. Для вычисления условной вероятности используется следующая формула:

$$P(d \mid s) = \frac{P(d \wedge s)}{P(s)}.$$

Как видно из данной формулы, условная вероятность определяется через понятие совместности или одновременности событий, то есть вероятности совпадения событий d и s, разделенное на вероятность события s. Из приведенной формулы очевидно, что вероятность совпадения или произведения двух событий равна произведению вероятности одного из них и условной вероятности другого, вычисленную при условии, что первое имело место:

$$P(d \wedge s) = P(s) \cdot P(d \mid s) \text{ или}$$
$$P(d \wedge s) = P(d) \cdot P(s \mid d).$$

Подставляя последнюю формулу в определение условной вероятности можно получить правило Байеса в простейшем виде:

$$P(d \mid s) = \frac{P(s \mid d) \cdot P(d)}{P(s)}.$$

Это правило позволяет определить вероятность $P(d \mid s)$ появления события d при условии, что произошло событие s через заранее известную условную вероятность $P(s \mid d)$. В полученном выражении $P(d)$ – априорная вероятность наступления события d, а $P(d \mid s)$ – апостериорная вероятность, то есть вероятность того, что событие d произойдет, если известно, что событие s свершилось. Данное правило иногда называют *инверсной формулой для условной вероятности*, так как она позволяет вычислить вероятность $P(d \mid s)$ через $P(s \mid d)$.

Для систем, основанных на знаниях, правило Байеса гораздо удобнее формулы определения условной вероятности через вероятность одновременного наступления событий $P(d \wedge s)$. В этом достаточно просто убедится. Пусть у пациента X имеется некоторый симптом *симптом_Y* и необходимо узнать, какова вероятность того, что этот симптом является следствием заболевания *заболевание_Z*. Для того чтобы непосредственно вычислить $P($*заболевание_Z* $\mid$ *симптом_Y*$)$, нужно оценить каким либо образом, сколько человек в мире страдают этим заболеванием, и сколько человек одновременно имеют *заболевание_Z* и *симптом_Y*. Такая информация, как правило, недоступна или отсутствует вообще. Особенно, что касается вычисления $P($*заболевание_Z* $\wedge$ *симптом_Y*$)$.

Однако, если посмотреть на вероятность не как на объективную частотность событий при достаточно долгих независимых испытаниях, а как на субъективную оценку совместного наступления событий, то ситуация значительно упрощается. Например, врач может не знать или не иметь возможности определить, какая часть пациентов имеющих *симптом_Y* страдают от *заболевание_Z*. Но на основании, например собственного опыта или литературных данных, врач в состоянии оценить у какой части пациентов имеющих *заболевание_Z* встречается *симптом_Y*. Следовательно, можно вычислить $P($*симптом_Y* $\mid$ *заболевание_Z*$)$ и применить инверсную формулу для условной вероятности.

Ситуация значительно усложняется, если речь пойдет о множестве симптомов и множестве заболеваний. Если необходимо вычислить условную вероятность для одного симптома из некоторого множества симптомов, то потребуется $m \cdot n + m + n$ вычислений, где m – количество возможных диагнозов, а n – число разнообразных симптомов. Учитывая, что в медицинской диагностике используются тысячи видов заболеваний и огромное количество симптомов, эта задача становится нетривиальной.

Ситуация еще усложниться, если включить в процесс постановки диагноза сразу несколько симптомов. Правило Байеса в обобщенной форме выглядит следующим образом:

$$P(d \mid s_1 \wedge ... \wedge s_k) = \frac{P(s_1 \wedge ... \wedge s_k \mid d) \cdot P(d)}{P(s_1 \wedge ... \wedge s_k)}.$$

Данная формула требует $(m \cdot n)^k + m + n^k$ вычислений оценок вероятностей, что даже при небольшом k является большим числом. Такое количество оценок требуется по той причине, что для вычисления $P(s_1 \wedge \ldots \wedge s_k)$ в общем случае сначала требуется вычислить $P(s_1 | s_2 \wedge \ldots \wedge s_k) \cdot P(s_2 | s_3 \wedge \ldots \wedge s_k) \cdot \ldots \cdot P(s_k)$. Однако если предположить что симптомы независимы, то количество вычислений резко снижается и становиться таким же, как и в случае учета единственного симптома, так как в для независимых s_i и s_j $P(s_i \wedge s_j) = P(s_i) \cdot P(s_j)$. Даже, если в действительности установить независимость невозможно, можно предположить наличие так называемой условной независимости. Это предположение может основываться на каких-либо фоновых знаниях. Например, если в автомобиле нет бензина и не работает свет, то исходя из общих представлений о конструкции автомобиля можно предположить, что эти симптомы независимы. Но если автомобиль не заводится и не работает освещение, то такое предположение сделать уже нельзя. Соответственно, в системе, использующей вероятностные оценки достоверности, необходимо предусмотреть средства отслеживания зависимости между используемыми данными.

4.3. Нечеткие множества и нечеткая логика

Эксперты при формировании оценок тех или иных признаков, симптомов или ситуаций, как правило, используют знания, основанные не на информации о конкретных примерах объектов, данных, отношений, а оперируют скорее понятиями классов объектов, отношений, гипотез и пр. Методы решений задач, таким образом, должны включать этап классификации данных или знаний. То есть конкретные экземпляры объектов, сигналов и т.п. рассматриваются как представители более общих классов, категорий. Но в реальных ситуациях редко встречаются объекты, которые точно соответствуют той или иной категории или классу. У конкретного экземпляра часть признаков может присутствовать, а другая часть отсутствовать. Таким образом, принадлежность этого объекта к какому-либо классу является размытой. Для формирования суждений о подобных категориях и принадлежащих к ним объектов был предложен формализм *теории нечетких множеств* [8]. Эта теория легла в основу *нечеткой логики*, позволяющей использовать понятие неопределенности в логических вычислениях.

Классическая теория множеств базируется на булевой, двухзначной логике. Принадлежность объекта к классу $a \in A$ может принимать значения *ИСТИНА*, если объект *a* входит в множество *A*, или *ЛОЖЬ* – в противоположном случае. После появления понятия «нечеткие множества», обычные множества стали также называть «жесткими». Именно присущая классической теории множеств «жесткость» при определении категорий, явилась источником проблем, при попытке применить ее для описания нечетко определенных категорий.

Суть теории нечетких множеств лучше всего рассмотреть на примере. Возьмем в качестве нечеткой категории понятие «быстрый». Если применить его к автомобилям, то тогда возникает вопрос: какой автомобиль можно считать быстрым. В классической теории множество *A* «быстрых автомобилей» можно сформировать либо перечислением конкретных представителей данного класса, либо введя в рассмотрение характеристическую функцию *f*, такую, что для любого объекта *X*

$f(X)$ = *ИСТИНА* тогда и только тогда, когда $X \in A$.

Например, эта функция может отбирать только те автомобили, которые могут развивать скорость более 150 км в час:

$$GT150(X) = \begin{cases} \textit{ИСТИНА, если } CAR(X) \textit{ и } MAX\_SPEED(X) > 150 \\ \textit{ЛОЖЬ, в противном случае} \end{cases}.$$

Эта функция, используя предикат $CAR(X)$ и функцию $MAX\_SPEED(X)$ представляет множество:

$$\{ X \in CAR \mid MAX\_SPEED(X) > 150 \}.$$

Эта формула утверждает, что элементами нового множества являются только те элементы множества *CAR*, для которых максимальная скорость превышает 150 км в час.

Представляя все множество «быстрых» автомобилей, интуитивно кажется, что границы этого множества должны быть размыты, а принадлежность элементов этому множеству может быть каким либо образом ранжирована (см. Рисунок 4-1. Нечеткое множество быстрых автомобилей.).

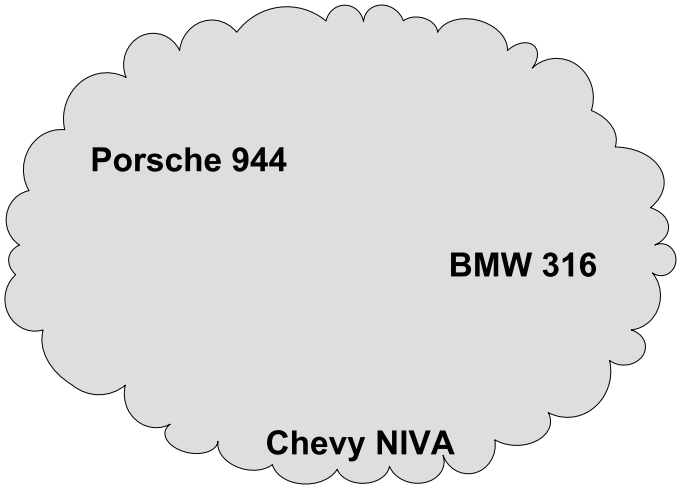


Рисунок 4-1. Нечеткое множество быстрых автомобилей.

Можно говорить, что каждый элемент (автомобиль) множества «быстрых» автомобилей более или менее типичен для данной категории. Следовательно, с помощью некоторой функции можно выразить степень принадлежность элемента к множеству. Пусть функция $f(X)$ определена на интервале [0, 1]. Тогда, если для объекта X функция $f(X) = 1$, то этот объект *определенно является членом множества*, а если для него $f(X) = 0$, то он *определенно не является членом множества*. Все промежуточные значения $f(X)$ выражают *степень принадлежности* к множеству. В примере с автомобилями требуется функция, оперирующая со скоростью. Ее можно определить таким образом, что $f_{FAST}(80) = 0$ и $f_{FAST}(180) = 1$, а все промежуточные значения представляются некоторой монотонной кривой, имеющей значения в интервале [0, 1] (см. Рисунок 4-2. Функция от скорости для понятия "быстрый".).

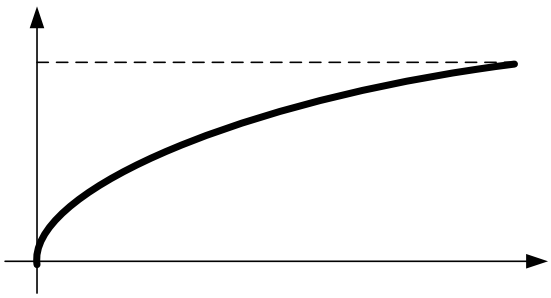


Рисунок 4-2. Функция от скорости для понятия "быстрый".

Для определения множества *FAST_CAR* «быстрых» автомобилей, на основании приведенной выше функции можно ввести новую характеристическую функцию, определенную на множестве всех автомобилей:

$$f_{FAST\_CAR}(X) = f_{FAST}(MAX\_SPEED(X)).$$

Членами этого множества, таким образом, становятся пары (*объект*, *степень*), например:

FAST_CAR = {(Porshe 944, 0,9), (BMW 316, 0,6), (Chevy NIVA, 0,1)}.

Для вычисления значений сложных выражений принадлежности элементов к множеству в классической теории множеств используется булева логика. Для нечетких множеств, принадлежность к которым выражается функцией, принимающей значения от 0 до 1, была создана *нечеткая логика*. Аппарат этой логики включает операции отрицания, конъюнкции, дизъюнкции и пр., учитывающие концепцию неопределенности. Например, логическое отрицание, по аналогии с теорией вероятности, реализуется в виде выражения: $\neg F(X) = 1 - F(X)$, где F – нечеткий предикат (нечеткая функция).

Аналоги операций конъюнкции и дизъюнкции в нечеткой логике не связаны с теорией вероятности и имеют следующие определения:

$$f_{F \wedge G}(X) = \min(f_F(X), f_G(X)),$$
$$f_{F \vee G}(X) = \max(f_F(X), f_G(X)).$$

Например, фраза «Porshe 944 является быстрым и представительским автомобилем» может быть представлена с помощью предикатов *FAST_CAR*(Porshe-944) и *PRETENTIOUS_CAR*(Porshe-944). Значение *FAST_CAR*(Porshe-944) = 0,9, а значение *PRETENTIOUS_CAR*(Porshe-944) пусть будет равно 0,7. Соответственно, можно вычислить значение конъюнкции этих предикатов:

$$FAST\_CAR(\text{Porshe-944}) \wedge PRETENTIOUS\_CAR(\text{Porshe-944}) =$$
$$= \min(0{,}9,\ 0{,}7) = 0{,}7.$$

Интересным следствием из приведенных выше определений операций нечеткой логики является следующий пример:

$$FAST\_CAR(\text{Porshe-944}) \wedge \neg FAST\_CAR(\text{Porshe-944}) = 0{,}1.$$

Очевидно, значение условной вероятности того, что это утверждение является истинным равно нулю:

$$P(FAST\_CAR(\text{Porshe-944}) \mid \neg FAST\_CAR(\text{Porshe-944})) = 0.$$

Однако в нечеткой логике это выражение вычисляется как min(0,9, 1 - 0,9) = 0,1. Смысл того, что это значение не равно нулю как раз и заключается в нечеткости понятия «быстрый». Кроме «быстрых» автомобилей существуют также «медленные» и «средние». Выражение *FAST_CAR*(Porshe-944) = 0,9 означает, что можно только с 90% уверенностью отнести этот автомобиль к категории «быстрый», так как существуют еще, например, и более быстрые автомобили, принимающие участия в соревнованиях Формулы 1. Поэтому полученное значение 0,1 говорит о том, что Porshe-944 принадлежит также и категории «среднескоростных» автомобилей, которые в чем-то близки к «быстрым», а в чем-то к «медленным».

Операторы нечеткой логики обладают свойствами коммутативности, ассоциативности и взаимной дистрибутивности. Как к операторам в стандартной логике, к ним применим принцип *композитивности*, то есть значения составных выражений вычисляются только по значениям входящих в них подвыражений. В этом операторы нечеткой логики составляют полную противоположность законам теории вероятностей, согласно которым при вычислении вероятности конъюнкции и дизъюнкции величин нужно принимать во внимание условные вероятности.

Одним из направлений в нечеткой логике является *теория возможности*, рассматривающая точно поставленные вопросы на нечетко сформулированных знаниях. Примером такого вопроса является утверждение «Вероятно, что X связан с Y». Существование объектов X и Y не вызывает сомнений, а вот наличие между ними связи ставится под вопрос.

Основные идеи теории возможности лучше всего рассмотреть на примере. Предположим, что в ящике находится 10 шаров, но известно, что только несколько из них являются красными. Какова вероятность того, что из ящика будет извлечен красный шар? Непосредственно вычислить эту вероятность невозможно, так как нам известно, только, что красных шаров несколько, а сколько именно не задано. Тем не менее, для каждого значения вероятности $P(RED)$ того, что шар является красным можно определить некоторое значение в диапазоне [0, 1]. Для этого следует определить понятие «несколько», как нечеткое множество, например:
$f_{SEVERAL}$ = {(2, 0.1), (3, 0.2), (4, 0.6), (5, 1.0), (6, 1.0), (7, 0.6), (8, 0.3) , (9, 0.1)}.

Из этого определения следует, что 3 из 10 с небольшой вероятностью можно считать синонимом «несколько», так как $f_{SEVERAL}(3) = 0.2$. Числа 5, 6 полностью согласуются с термином «несколько», а числа 1 и 10 не рассматриваются вовсе, как соответствующие этому понятию. Нечеткое множество, определенное на множестве чисел получило название *нечеткие числа*. Аналогичным образом можно определить нечеткие множества для понятий «мало», «почти» и пр.

Распределение возможностей теперь можно вычислить по формуле

$$f_{P(RED)} = \frac{f_{SEVERAL}}{10}.$$

Подставляя в данную формулу множество, определенное выше получается новое множество:
{(0.2,0.1), (0.3,0.2), (0.4,0.6), (0.5,1.0), (0.6,1.0), (0.7,0.6), (0.8,0.3) , (0.9,0.1)}.

Таким образом, возможность того, что $P(RED) = 0.3$ составляет 20%. Множество $f_{P(RED)}$ также называют *нечеткой вероятностью*.

С помощью данного механизма можно ввести в рассмотрение любую функцию, поэтому удобно использовать понятие *нечеткого правдоподобия*. Например, некоторое утверждение может быть оценено

как «очень правдоподобное» или «частично правдоподобное». Эти понятия представляются на множестве, где областью определения и областью значений функции являются значения правдоподобия в нечеткой логике f_{TRUE}: [0, 1]→[0, 1]. Следовательно, можно рассматривать значения, меньшие единицы, как «достаточно правдоподобные». Например, для ситуации с быстрыми автомобилями можно определить

$$TRUE(FAST\_CAR(\text{Porsche-944})) = 1.$$

Таким образом, можно с полной уверенностью утверждать, что Porsche-944 является быстрым автомобилем, несмотря на то, что на рынке имеются и более скоростные автомобили.

4.4. Теория Демпстера-Шефера

Одним из ограничений вероятностных подходов к неопределенности является то, что они используют единственную количественную меру неопределенности, вычисление которой может оказаться очень сложной задачей. Альтернативный подход, называемый *теорией обоснования Демпстера-Шефера* рассматривает множества предположений (гипотез) и ставит в соответствие каждому из них *вероятностный интервал неопределенности* (правдоподобия), которому принадлежит степень уверенности в каждом предположении. Кроме того, этот подход решает проблему измерения достоверности, делая коренное различие между отсутствием уверенности и незнанием.

Теория Демпстера-Шефера предлагает две основные идеи: средства вычисления функций доверия (*belief function*) на таких множествах гипотез и правила объединения нескольких функций доверия, сформулированных на основании разных свидетельств. По мере накопления информации свидетельствующей в пользу какой либо из конкурирующих гипотез интервалы неопределенности уменьшаются, а степень доверия к гипотезе – увеличивается.

В теории Демпстера-Шефера (Dempster-Shafer) предполагается, что описание некоторой предметной области представляет собой пространство Θ взаимно исключающих гипотез. Множество Θ должно быть исчерпывающим, то есть набор входящих в Θ альтернативных гипотез должен полностью описывать все аспекты предметной области. В терминах данной теории пространство гипотез получило название *различающий фрейм* (*frame of discernment*). Примером Θ может быть множество всех возможных заболеваний в задачах постановки диагноза.

Также предполагается, что мы располагаем средством получения свидетельств как в пользу отдельных гипотез $h_1,..., h_n$, принадлежащих Θ, так и в пользу целых подмножеств гипотез $A_1,..., A_k$, которые могут

пересекаться. Все эти свидетельства можно рассматривать как элементы множества Ψ. Если в множестве Θ содержится n гипотез, то количество подмножеств Θ равно 2^n. Следовательно, можно построить отображение

$$\Gamma: \Psi \rightarrow 2^{\Theta},$$

которое будет связывать каждый элемент в Ψ с подмножеством пространства Θ. Каждое такое подмножество называется *фокальным элементом*. Отметим, что предположение об исчерпывающей полноте набора гипотез означает, что ни один из элементов $\psi \in \Psi$ не отображается на пустое множество. Другими словами, для любого свидетельства существует хотя бы одна гипотеза, достоверность которой подтверждает это свидетельство. Например, свидетельство «Заболевание является инфекционным» соответствует подмножеству элементов из Θ, которые представляют инфекции {«грипп», «гепатит», …}.

Для определения начальных значений достоверности свидетельств используется функция *присвоения базовых вероятностей* $m()$ (*bpa – basic probability assignment*), которая определена на множестве 2^{Θ} значений из интервала [0,1], такая, что

$m(\varnothing) = 0$ и

$\sum(m(A_i)) = 1$, где суммирование выполняется по всем $A_i \in 2^{\Theta}$.

Величина $m(A_i)$ представляет степень достоверности свидетельства, уверенность в истинности гипотез, подтвержденных свидетельством A_i. Например, если $m(A) = 0{,}4$, то уверенность в A составляет 40% (но не в каком-либо подмножестве A).

Значение суммарного доверия (*функция доверия – belief function*, обозначаемая *Bel*) для любого фокального элемента A может быть найдено суммированием значений m по всем подмножествам в A. Таким образом, *Bel* является функцией, определенной на множестве 2^{Θ} значений из интервала [0,1], такой, что

$Bel\ (\varnothing) = 0$,

$Bel\ (\Theta) = 1$,

для любого $A \subseteq \Theta$, $Bel(A) = \sum_{B \subseteq A} m(B)$, где различные B являются подмножествами в A.

Функция доверия присваивает каждому подмножеству в Θ величину полного доверия к свидетельству, представленному данным подмножеством.

Для каждого подмножества A можно ввести, также, *функцию сомнения* (*doubt function*, обозначаемую *Dou*), которая выражает доверие к дополнению множества A или, иначе говоря, доверие к *не A*

$Dou\ (A) = Bel\ (\neg A)$, где $\neg A$ – дополнение A.

Оценка привлекательности A (обозначаемая $Pls(A)$), представляет степень совместимости свидетельства с гипотезами входящими в A и может быть вычислена по формуле

$Pls(A) = \sum_{B \cap A \neq \varnothing} m(B)$, где различные B являются подмножествами в A.

Поскольку определенная таким образом оценка привлекательности A есть не что иное, как мера нашего недоверия к $\neg A$, то можно записать:

$$Pls(A) = 1 - Bel(\neg A) = 1 - Dou(A).$$

Истинная оценка доверия к фокальному элементу A будет ограничена снизу *оценкой доверия* к A, а сверху – *оценкой привлекательности A*

$$[Bel(A), Pls(A)].$$

Значение оценки привлекательности A можно рассматривать как предел, до которого можно улучшить гипотезы из A при наличии свидетельств а пользу гипотез-конкурентов. Удобно рассматривать информацию, содержащуюся в оценке *Be1* для данного подмножества, в виде *доверительного интервала* в форме [*Be1*(A), *Pls*(A)]. Ширина интервала может служить оценкой неуверенности в справедливости гипотез из A при имеющемся наборе свидетельств.

Правило Демпстера позволяет вычислить новое значение функции доверия по двум значениям, базирующимся на разных наблюдениях. Обозначим Bel_1 и Bel_2 два значения функции доверия, которым соответствуют два значения функции присвоения базовых вероятностей m_1 и m_2. Правило позволяет вычислить новое значение $m_1 \oplus m_2$, а затем и новое значение функции доверия $Bel_1 \oplus Bel_2$, основываясь на определениях, приведенных выше.

Для гипотезы A значение $m(A) = m_1 \oplus m_2(A)$ есть ортогональная сумма всех произведений в форме $m_1(X) \cdot m_2(Y)$, где X и Y подмножества в Θ, пересечением которых является A.

Если в таблице пересечений будет обнаружен пустой элемент, выполняется нормализация. В процедуре нормализации значение K определяется как сумма всех ненулевых значений, присвоенных в множестве Θ, затем $m_1 \oplus m_2$ ($\emptyset$) присваивается значение ноль, а значения $m_1 \oplus m_2$ для всех других множеств гипотез делится на $(1 - K)$.

Таким образом,

$$m_1 \oplus m_2(A) = \frac{\sum_{X \cap Y = A} m_1(X) \cdot m_2(Y)}{1 - \sum_{X \cap Y = \emptyset} m_1(X) \cdot m_2(Y)},$$

или, изменяя утверждение в знаменателе на противоположное

$$m_1 \oplus m_2(A) = \frac{\sum_{X \cap Y = A} m_1(X) \cdot m_2(Y)}{\sum_{X \cap Y \neq \emptyset} m_1(X) \cdot m_2(Y)}.$$

Следует учитывать, что значения m_1 и m_2 сформированы по независимым источникам свидетельств в пределах того же пространства гипотез. Обратите внимание и на тот факт, что вследствие коммутативности операции умножения правило Демпстера дает один и тот же результат при любом порядке объединения свидетельств. Если значение $(1 - K) = 0$, то ортогональная сумма $m_1 \oplus m_2$ не существует. В

таком случае говорят, что m_1 и m_2 *полностью противоречивы*. Мерой противоречия между Bel_1 и Bel_2 является оценка конфликта (*weight of conflict*, обозначается *Con*)

$$Con\ (Bel_1, Bel_2) = Log(1 - K)^{-1}.$$

В случае, когда A является пересечением многих (более двух) подмножеств формула ортогональной суммы принимает вид

$$m(A) = \frac{\sum_{\cap A_i = A} \prod_{i=[1,n]} m_i(A_i)}{1 - \sum_{\cap A_i = \emptyset} \prod_{i=[1,n]} m_i(A_i)}.$$

Рассмотрим следующий пример, иллюстрирующий применение приведенных выше формул. Допустим, Θ есть множество гипотез{H, C, P}. Пусть, также, заданы базовые вероятности

m({H}) = 0,3
m({H, C}) = 0,2
m({H, C, P}) = 0,5

Сумма всех вероятностей, как это и должно быть, равна 1. Доверие $Bel(A)$ к произвольному подмножеству A в множестве Θ вычисляется сложением всех $m(B)$, где B есть подмножество A. Следовательно, функции доверия для данного примера могут вычисляться следующим образом

Bel ({H}) = m({H}) = 0,3
Bel ({H, C}) = m({H}) + m({H, C}) = 0,3 + 0,2 = 0,5
Bel ({H, P}) = m({H}) = 0,3
Bel ({H, C, P}) = m({H}) + m({H, C}) + m({H, C, P}) = 0,3+0,2+0,5

Соответственно, функции сомнения будут следующими

Dou ({H}) = *Bel* ({C, P}) = 0
Dou ({H, C}) = *Bel* ({P}) = 0
Dou ({H, P}) = *Bel* ({C}) = 0
Dou ({H, C, P}) = *Bel* (Ø) = 0

Наконец, оценки привлекательности, показывающие верхние оценки вероятности, следующие

Pls ({H}) = 1 - *Dou* ({H}) = 1
Pls ({H, C}) = 1 - *Dou* ({H, C}) = 1
Pls ({H, P}) = 1 - *Dou* ({H, P}) = 1
Pls ({H, C, P}) = 1 - *Dou* ({H, C, P}) = 1

Поскольку комбинирование функций доверия более объемная задача, рассмотрим ее на более простой предметной области Θ` = {D, D`}

m_1({D}) = 0,8 m_2({D}) = 0,9
m_1({D`}) = 0 $m_2$({D`}) = 0
m_1{D, D`}) = 0,2 $m_2$({D, D`}) = 0,1

Для вычисления ортогональной суммы построим таблицу произведений функций доверия для всех подмножеств Θ`

		m_2		
		{D} : 0,9	{D`} : 0	{D, D`} : 0,1
m_1	{D} : 0,8	0,72	0	0,08
	{D`} : 0	0	0	0
	{D, D`} : 0,2	0,18	0	0,02

Далее вычисляем $K = \sum_{X \cap Y = \varnothing} m_1(X) \cdot m_2(Y)$. В данном примере есть только два случая, когда $X \cap Y = \varnothing$

$m_1(\{D\}) \cap m_2(\{D`\})$ и $m_1(\{D`\}) \cap m_2(\{D\})$.

Из таблицы видно, что соответствующие значения произведений равны нулю. Следовательно, $K = (0 + 0) = 0$. Окончательно имеем

$m_1 \oplus m_2(\{D\}) = (0{,}72 + 0{,}08 + 0{,}18) / (1 - 0) = 0{,}98$

$m_1 \oplus m_2(\{D`\}) = (0) / (1 - 0) = 0$

$m_1 \oplus m_2(\{D, D`\}) = (0{,}02) / (1 - 0) = 0{,}02$

Таким образом, на основании заданных функций доверия m_1 и m_2 можно утверждать, что наиболее вероятное утверждение для данной предметной области – D. Оценка конфликта для данного примера $Log(1) = 0$. То есть можно утверждать, что m_1 и m_2 не противоречивы.

Рассмотрим теперь работу теории Демпстера-Шефера на более конкретном примере, основанном на привычных высказываниях. Предположим, моя коллега Мелисса говорит мне, что мой компьютер сломался. Пусть вероятность того, что ей можно верить, составляет 0,9, а того, что верить нельзя – 0,1. Утверждение Мелиссы истинно, если ей можно верить, но оно не обязательно ложно, если ей верить нельзя (в соответствии с одним из основных положений теории, разделяющих неуверенность и незнание). Таким образом, утверждение Мелиссы о том, что мой компьютер сломался обосновывается с достоверностью 0,9, а то, что он исправен – с достоверностью 0 (так как нет свидетельств в пользу этой гипотезы). Мера правдоподобия будет равна

$$Pls(\textit{сломался}) = 1 - Bel(\neg\ \textit{сломался}) = 1 - 0{,}0 = 1.$$

Таким образом, мера доверия к Словам Мелиссы есть [0,9; 1]. Но, пока еще нет оснований считать, что компьютер не сломался.

Теперь, предположим, что другой коллега Билл также говорит, что мой компьютер сломался. Пусть вероятность того, что Биллу можно верить составляет 0,8, а что верить нельзя – 0,2. И пусть, также, утверждения Мелиссы и Билла о моем компьютере независимы друг от друга, то есть, вызваны разными причинами. Утверждение «Билл заслуживает доверия» также должно быть независимым от степени доверия к Мелиссе, тогда можно будет вычислить вероятность правдивости обоих утверждений как произведение их вероятностей – $0{,}8 \cdot 0{,}9 = 0{,}72$. Вероятность сомнительности обоих утверждений – $0{,}2 \cdot 0{,}1 = 0{,}02$. А вероятность того, что верить можно, по крайней мере, одному из

них (мера правдоподобия) – 1 - 0,02 = 0,98. Следовательно, на основании вероятностей обоих утверждений о том, что компьютер сломался, можно вычислить степень достоверности этого события [0,98; 1].

Теперь, посмотрим, что будет, если Билл и Мелисса говоря противоположные вещи: Мелисса утверждает, что компьютер сломан, а Билл говорит, что нет. В этой ситуации они оба не могут говорить правду и, следовательно, не могут вызывать доверие. Возможны две ситуации: либо нельзя верить никому, либо только одному из них. Априорная вероятность того, что можно верить только Мелиссе, а Биллу нет – $0,9 \cdot 0,2 = 0,18$. А того, что верить можно только Биллу, а Мелиссе нет – $0,8 \cdot 0,1 = 0,08$. Наконец, вероятность того, что верить нельзя никому – $0,2 \cdot 0,1 = 0,02$. Имея вероятность того, что по крайней мере одному из коллег верить нельзя – 0,18 + 0,08 + 0,02 = 0,28 –, можно вычислить апостериорную вероятность того, что верить можно лишь Мелиссе и мой компьютер сломан – 0,18 / 0,28 = 0,643; или апостериорную вероятность того, что прав Билл, и мой компьютер работает – 0,18 / 0,28 = 0,286. Правдоподобие поломки в этой ситуации составляет

$$Pls(\textit{сломался}) = 1 - Bel(\neg\, \textit{сломался}) = 1 - 0,286 = 0,714.$$

Таким образом, достоверность утверждения, что компьютер сломался, лежит в интервале [0,643; 0,714].

Примечательно, что в вычислениях использовалась вторая форма правила Демпстера. Аналогичные результаты можно получить при использовании основной формы этого правила. Вместо сложения возможных произведений вероятностей в знаменателе, можно сложить произведения вероятности гипотез, пересечение которых дает пустое множество, и затем вычесть полученную сумму из единицы. Как уже было отмечено, невозможной является единственная ситуация, когда оба говорят правду. Следовательно, в знаменателе правила объединения свидетельств должно быть значение $1 - 0,8 \cdot 0,9 = 1 - 0,72 = 0,28$ – то что получилось при сложении произведений вероятностей для трех допустимых ситуаций.

4.5. Вопросы для самопроверки и упражнения

1. Назовите основные источники неопределенности знаний встречающиеся при разработке экспертных систем.

2. Каким образом применяются коэффициенты уверенности в системе MYCIN? По какому закону комбинируются коэффициенты уверенности для различных правил?

3. Как можно представить неопределенность знаний с помощью правила Байеса? Какова, при этом, вычислительная сложность получения оценки вероятности?

4. В чем заключается отличие нечетких множеств от обычных «жестких» множеств? Приведите пример описания нечеткого понятия и множества, построенного на этом понятии.

5. Как определяются нечеткие операции «отрицание», «логическое и» и «логическое или» в нечеткой логике?

6. Изложите основную идею теории возможности. Приведите пример описания нечеткой вероятности.

7. Приведите пример множества гипотез, функций доверия и сомнения, а также оценки привлекательности для этого множества в соответствии с положениями теории обоснования Демпстера-Шефера.

8. Вычислите по правилу Демпстера значение функции доверия для множества $\Theta = \{A, B, C\}$, если известны значения функций базовых вероятностей для двух различных наблюдений:

$m_1(\{A\}) = 0{,}3$	$m_2(\{A\}) = 0{,}2$
$m_1(\{A, B\}) = 0{,}4$	$m_2(\{A, B\}) = 0$
$m_1(\{A, B, C\}) = 0{,}3$	$m_2(\{A, B, C\}) = 0{,}8$

9. Объясните различие между понятиями «неуверенности» и «незнания» с точки зрения теории Демпстера-Шефера.

10. Придумайте несколько прикладных областей, в которых необходимы рассуждения в условиях неопределенности. Приведите возможные правила (знания) для данной области, использующие нечеткие знания с использованием коэффициентов уверенности, вероятностей и нечетких множеств.

Литература

1. Н. Винер “Кибернетика”, М.: Наука, 1983.

2. Альтшуллер Г.С., Злотин Б.Л., Зусман А.В., Филатов В.И. Поиск новых идей: от озарения к технологии. Кишинёв: Картя молдовеняскэ, 1989.

3. Перспективы развития вычислительной техники. Интеллектуализация ЭВМ. М.: Высшая школа, 1989.

4. Гаврилова Т.А. Базы знаний интеллектуальных систем. СПб: Питер, 2001.

5. Дж. Элти, М. Кумбис Экспертные системы: концепции о примеры. М.: Финансы и статистика, 1987.

6 . Минский М.Л. Фреймы для представления знаний. М.: Энергия, 1979.

7 . Искусственный интеллект. Стратегии и методы решения сложных проблем. Джорж Ф. Люггер. СПб: Вильямс, 2003.

8 . А.Кофман Введение в теорию нечетких множеств. М.: Радио и связь, 1982.

9. Экспертные системы. Принципы работы и примеры. Под ред. Р. Форсайта. М.: Радио и связь, 1987.

10. Питер Джексон Введение в экспертные системы. М.: Вильямс, 2001.

Кафедра проектиования компьютерных систем была образована в сентябре 1945 года как подразделение нового факультета электроприборостроения (позднее в 1952г. Переименованного в радиотехнический). Кафедра стала готовить инженеров, специализирующихся в новых направлениях радиоэлектронной техники, таких как радиолокация, радиоуправление, теленаведение и др. Организатором и первым заведующим кафедрой был профессор Зилитинкевич С. И., который является крупнейшим деятелем в области радиотехники и электроники, автором ряда важнейших исследований и открытий. Наиболее выдающимся из них били обнаружение «собственных колебаний электронов» в электронных лампах. На этой основе им были получены впервые электромагнитные волны дециметрового диапазона, что явилось важнейшим вкладом в возникновение и развитие современной техники сверх высоких частот. Название кафедры в тот период открыто не упоминалось, а она имела номер 11.

Профессор Зилитинкевич С. И. Руководил кафедрой с 1945 оп 1951 год. Коллектив кафедры номер 11 формировался несколько лет. Первыми её преподавателями были А. Н. Иванов и А. И. Сапетин. Оба пришли в институт после демобилизации из армии, оба защищали Ленинград. А. Н. Иванов с первых дней войны был в штабе ПВО города, работал на первых отечественных РЛС. Он относился к числу тех специалистов, которые заставили поверить руководство фронта в надёжность и эффективность нового средства наблюдения за самолётами противника вместо распространённого тогда метода звукоулавливания. Первым штатным преподавателем был выпускник ЛИТМО 1945 года, прошедший специальную подготовку К. Е. Медведев - будущий доцент и декан факультета.

В течение первого года своего существования кафедра развивалась чрезвычайно быстро. Из вновь привлечённых преподавателей прежде всего седует отметить профессоров Б. А. Остроумова и Л. Б. Смияна. С 1946 года к работе на кафедре № 11 приступили старший преподаватель А. А. Тударовский, ассистенты Л. А. Гирелик, К. Г. Шаров, старшие лаборанты Г. В. Метр и П. Л. Косьмин. Основной лабораторной базой в то время были радиолокационные станции типа <Вюрсбург> (снятая с немецкого поезда) и первая отечественная станция типа <Пегматит>. Лишь в пятидесятые годы появились на кафедре действующие отечественные станции <Мист-2>, <Кобальт>, <П-8> и ряд других.

С 1951 года по 1954 кафедру возглавлял крупный специалист в области передающих устройств РЛС, один из ведущих работников

радиопромышленности, кандидат технических наук, доцент А. И. Лебедев - Карманов (по совместительству). В 1954 году А. А. Тударовский, ставший к этому времени доцентом был избран заведующим кафедры № 11.

Постепенно состав кафедры начал пополняться её молодыми выпускниками. Курс «Антенны и распределение радиоволн» стал читать старший преподаватель Н. Н. Филиппов, курс «Радиолокационные системы» – выпускник кафедры 1948 года, закончивший аспирантуру в ЛЭТИ, к. т. н. А. Н. Гарпина - Домченко, курс «радиоприёмные устройства» Б. Н. Меньшов, курс «Электропитание радиоустройств» Н. В. Ефимов. Одновременно на кафедру поступило новое оборудование, в том числе современная измерительная аппаратура, что позволило создать собственную лабораторную базу по всем курсам. Большую работу провели ассистент Л. В. Шенников, старшие лаборанты Э. С. Кочанов, Г. Н. Грязин, К. Г. Шаров, ставший затем заведующим лабораторией.

На кафедре № 11 проводилась также большая научно-исследовательская работа. Так, в 1952 - 1953 годах по заказу Военно-медицинской академии был разработан и изготовлен первый отечественный электокардиограф. В 1955 - 1957 годах под руководством заведующего кафедрой доцента А. А .Тудировского по заказу Танкового КБ проводились работы, связанные с созданием радиолокационной Техники 8 мм. диапазона.

В 1957 году кафедры № 11 и № 76 были реорганизованы. На их базе радиолокационных приборов и устройств (РЛПУ) и радиоприёмных и радиопередающих устройств (РППУ). Первую кафедру возглавил крупный специалист в области телевидения и авиационной радиолокации, заведующий лабораторией одного из ОКБ К.Н.Т. Б. С. Мишин. С приходом Б. С. Мишина стало развиваться телевизионное направление в деятельности кафедры. Следует отметить, что он занимался телевидением ещё в 30-е годы, будучи студентом Томского государственного университета и практикантом находившейся в Ленинграде центральной радиолаборатории. В конце 30-х годов он работал на заводе имени Козицкого, где был одним из создателей первого серийного телевизора Т-1 и организации его производства. В годы войны он стал заниматься радиолокационной тематикой. В это время на кафедре развивалась работа по организации учебной и научно-исследовательской лаборатории телевидения, в которой принимал участие старший преподаватель А. И. Сапетин, занимавшийся телевидением и фотографией ещё в предвоенные годы, ассистент Г. Н. Грядин и А. В. Краситкин. Позднее к ним присоединился старший лаборант В. М. Таукчи. Под руководством Б. С.

Мишина, на кафедре бала выполненная первая НИР по телевизионной тематике - разработали и изготоваили действующий макет подводной установки СТУ-1. Второй крупной работой было создание приёмо-передающей быстродействующей двутелеграфной аппаратуры с плоской развёрткой. Общее руководчтво этой НИР осущеситвляли профессор М. М. Русинов и доцент Б. С. Мишин. В 1965 году на кафедре был изготовлен телевизионный стробоскоп. Это привело к новому направлению в прикладном телевидении, получивших теоретическое обоснование в докторской диссертации Г. Н. Грядина. В это время состав кафедры пополнился доцентами И. В. Ивановым, И. Ю. Рагинским, В. А. Смирновым, старшим преподавателем А. А. Зелетенкиевичем и ассистентом Ю. Н. Пановым. Был принят на кафедру только что, окончивший аспирантуру при кафедре, выпускник этой же кафедры, кандидат технических наук В. С. Салтыков. Заведовали лабораторией Н. В. Александров и затем В. М. Лакунин. В качестве инженера НИСа бал принят А. В. Панков, впоследствии ставший доцентом, который только что закончил эту кафедру.

После ухода Б.С.Мишина в 1964 году пенсию, кафедру в течение года возглавлял пришедший из ВНИИ телевидения доцент И.П.Захаров. После скоропостижной кончины И.П. Захарова заведующим кафедрой был назначен доцент А.Н.Иванов.

В начале 60х годов в отечественной промышленности обнаружилась острая нехватка конструкторов и технологов радиоаппаратуры особенно специалистов в области микроминиатюризации. В связи с этим кафедра была переименована в кафедру «Конструирования и производства радиоэлектронной аппаратуры» и начала подготовку инженеров по этому направлению. Были введены новые курсы. Прежние курсы были сняты или существенно сокращены или изменены. Потребовалась коренная реорганизация лабораторной базы и переаттестация преподавательского состава. Новые курсы пришлось готовить и организовывать лабораторную базу ведущим доцентом кафедры Г.Н.Грязнину, В.С.Салтыкову и В.А. Смирнову. Была организована научная работа по этому направлению , в частности был заключен большой договор с Петродворцовским часовым заводом на разработку электронного хронометра в микроминиатюрном исполнении. По окончании этой работы на данный хронометр были получены несколько авторских свидетельств.

В 1970 году радиотехнический факультет ЛИТМО был ликвидирован. Кафедру КиПРЭА переименовали в кафедру «Конструирования и производства электронно-вычислительной аппаратуры» (КиПВА) и перевели на факультет точной механики и

вычислительной техники. Коренной переделке подвергся переделке учебный план по которому велась подготовка специалистов. Были выдели два основных направления: автоматизация конструирования ЭВА и технология производства микроэлектронных устройств ЭВА.

В конце 1973 года на должность заведующего был избран д.т.н. профессор Ф.Г.Старос. Профессор Ф.Г.Старос являлся одним из основных родоначальников отечественной микроэлектроники. Он был главным разработчиком Советского центра микроэлектроники в Зеленограде. Под его руководством был разработан и издан первым в мировой практике прообраз персонального компьютера - УМ-1-НХ. В связи с назначением профессора Ф.Г.Староса директором одного из институтов А.Н.СССР во Владивостоке в начале 1974 года, он вынужден был покинуть кафедру КиПЭВА и на должность заведующего кафедрой был избран выпускник ЛИТМО 1959 года к.т.н. доцент В.В.Новиков (впоследствии д.т.н. профессор). С приходом В.В.Новикова резко усилилась работа в области микроэлектроники, были открыты научно-исследовательские темы по применению новых физических принципов при разработке различных электронных устройств. Большое участие в этих разработках принимали доценты А.В.Панков и В.С.Салтыков. В это время на фабрику был принят специалист в области схемотехники доцент В.Г.Фейгельс и ведущие специалисты промышленности, занимающиеся микроэлектроникой профессор Я.М.Беккер и доцент А.М.Скворцов (впоследствии профессор). Была налажена работа с промышленностью. Так, по договору с заводом «Россия» на территории кафедры был открыт промышленный участок по разработке и производству современных пленочных микросхем. Было установлено современное оборудование для вакуумного напыления, открыта линия по фотолитографии и т.д. На этом участке помимо решения чисто производственных задач, проводились лабораторные и научно-производственные работы для студентов и преподавателей кафедры, а сотрудники завода «Россия» работавшие на этом участке привлекались для работы на кафедре (руководство курсовым и дипломным проектированием, руководство лабораторными работами и др.).

С 1976 по 1996 кафедрой руководил известный специалист в области автоматизации проектирования электронных устройств профессор Г.А.Петухов (с небольшим перерывом когда с 1988 по 1992 год кафедру возглавлял ученик Г.А.Петухова профессор С.А.Арустамов, который в дальнейшем ушел из ЛИТМО , в связи с переходом на другую работу). За это время из других подразделений института доцент А.Л.Кузнецов и к.т.н. С.Ю.Яковлева. Получило дальнейшее направление развитие автоматизации проектирования. Был создан один из первых в ЛИТМО

собственный машинный класс. Научная работа была в основном сконцентрирована в области САПР. Так в это время на кафедре проводилась большая научно-исследовательская работа по автоматизации топологического проектирования БИС на базовых кристаллах, которую возглавляли профессора С.А.Арустамов и Г.А.Петухов.

В эти годы коллектив преподавателей пополнялся ее выпускниками: Н.С.Кармановским, Ю.А.Гатчиным, Б.А.Крыловым, К.О.Ткачевым, В.А.Ткалич, Е.К.Фролковой. Пришла на кафедру выпускница ЛИАПа Н.Ю.Иванова, а в 1988 году кафедра была переименована в кафедру микроэлектроники и автоматизации проектирования (МАП).

С1996 года кафедру возглавляет ее воспитанник доцент Ю.А.Гатчин. Помимо традиционной подготовки инженеров конструкторов-технологов по микроэлектронике и автоматизации проектирования вычислительных средств (специальность 2205), была начата подготовка специалистов по специальности 0754 «Комплексная защита объектов информатизации», причем основное внимание уделяется программно-аппаратной защите информации компьютерных систем.

В 1998 году кафедра была переименована и получила название «Кафедра проектирования компьютерных систем», что отразило содержание основных научных исследований и направления подготовки студентов и аспирантов.

За прошедшие годы кафедра подготовила более 4000 дипломированных инженеров. Более 50 молодых ученых защищали кандидатские диссертации, 10 человек защитили диссертации на соискание ученой степени доктора наук. Выпускники кафедры работают в ведущих научных центрах и учебных заведения России, Европы, Азии и Америки в промышленных и коммерческих фирмах, лабораториях и кафедрах университета.

В настоящее время кафедра является одним из ведущих российских научных и образовательных центров, ориентированным на фундаментальные и прикладные исследования в области микроэлектроники и САПР, подготовку высококвалифицированных специалистов 21 века.

РЛПУ (1945-1966)

Решением Правительства в августе 1945 года в ЛИТМО был открыт факультет электроприборостроения. Приказом по Институту от 17 сентября 1945 года на этом факультете была организована кафедра радиолокационных приборов и устройств, которая стала готовить инженеров, специализирующихся в новых направлениях радиоэлектронной техники, таких как радиолокация, радиоуправление, теленаведение и др. Организатором и первым заведующим кафедрой был д.т.н. профессор ЗИЛИТИНКЕВИЧ С.И. (до 1951 года). Выпускникам кафедры присваивалась квалификация "инженер - радиомеханик", а с 1956 года - "радиоинженер" (специальность 0705). В разные годы заведовали кафедрой доцент МИШИН Б.С, доцент ЗАХАРОВ И.П., доцент ИВАНОВА А.Н.

КиПРЭА (1966-1970)

Кафедра конструирования и производства радиоэлектронной аппаратуры. Каждый учебный план специальности 0705 коренным образом отличался от предыдущих планов радиотехнической специальности своей четко выраженной конструкторско - технологической направленностью. Оканчивающим институт по этой специальности присваивалась квалификация "инженер конструктор - технолог РЭА". Заведовал кафедрой доцент ИВАНОВ А.Н.

КиПЭВА (1970-1988)

Бурное развитие электронной вычислительной техники и внедрение ее во все отрасли народного хозяйства потребовали от отечественной радиоэлектронной промышленности решения новых ответственных задач. Кафедра стала готовить инженеров по специальности 0648, Подготовка проводилась по двум направлениям: автоматизация конструирования ЭВА и технология микроэлектронных устройств ЭВА, Заведовали кафедрой д.т.н, проф. НОВИКОВ В.В. (до 1976 г.), затем проф. ПЕТУХОВ Г.А.

Кафедра МАП (1988-1997)

Кафедра микроэлектроники и автоматизации проектирования выпускала инженеров конструкторов - технологов по микроэлектронике и автоматизации проектирования вычислительных средств (специальность 2205). Выпускники этой кафедры имеют хорошую технологическую подготовку и успешно работают как в производстве полупроводниковых интегральных микросхем, так и при их проектировании, используя современные методы автоматизации проектирования. Инженеры специальности 2205 требуются микроэлектронной промышленности и предприятиям - разработчикам вычислительных систем.
Кафедрой с 1988 по 1992 год руководил профессор АРУСТАМОВ С.А., затем снова профессор ПЕТУХОВ Г.А.

Кафедра ПКС (с 1997)

Кафедра проектирования компьютерных систем выпускает инженеров по специальности "Проектирование и технология электронных средств". Область профессиональной деятельности выпускников включает в себя проектирование, конструирование и технологию электронных средств, отвечающих, целям их функционирования, требованиям надежности, дизайна и условиям эксплуатации, Кроме того, кафедра готовит специалистов по специальности 0754 "Комплексная защита объектов информации", причем основное внимание уделяется программно-аппаратной защите информации компьютерных систем, С 1996 года кафедрой заведует д.т.н., профессор ГАТЧИН Ю.А.
За время своего существования кафедра выпустила 4023 инженера, из них по специальности 0705 - 2472 чел., по специальности 0648 (2205) - 1551 чел. и по специальности 0648 220600 – 28 чел. На кафедре защищены более 50 кандидатских диссертаций и 8 докторских.