Российский Государственный университет нефти и газа им. И.М.Губкина

Л.И. Григорьев, О.А. Степанкина

Системы искусственного интеллекта

учебное пособие

Рекомендовано Методическим Советом Университета
в качестве учебного пособия для студентов специальности 22.02
«Автоматизированные системы обработки информации и управления"

Москва, 1998

УДК 681
Г 83

Григорьев Л.И., Степанкина О.А. Системы искусственного интеллекта: Учебное пособие. М.: РГУ нефти и газа им. И.М.Губкина,- 1998. 59 с.

Данное учебное пособие предназначено для студентов кафедры АСУ, специализирующихся в области искусственного интеллекта (220205). В нефтегазовой отрасли имеется значительное число плохо формализуемых задач, для решения которых требуется применение систем искусственного интеллекта. В работе рассматриваются основные направления развития систем искусственного интеллекта, модели представления знаний. Более подробно описывается представление знаний на основе логики предикатов и декларативный язык Пролог. Анализируются примеры использования и перспективы применения систем искусственного интеллекта в нефтегазовой отрасли.

Курсом “Системы искусственного интеллекта” в учебном плане кафедры АСУ начинается цикл дисциплин (экспертные системы, нейронные сети, технология распознавания нечетких объектов и др.) по интеллектуальным автоматизированным системам.

Рецензент - Берман Р.Я., к.т.н., ОАО «Газавтоматика»

Содержание

Введение

Искусственный интеллект (ИИ) вызывал интерес у исследователей еще задолго до нашего столетия. Но только с середины 20-го века, когда получили активное развитие средства вычислительной техники, стало возможным проведение фундаментальных и прикладных работ по ИИ. Большая часть этих работ выполнялась ранее и реализуется сейчас в рамках государственных и межгосударственных проектов и программ. Использование достижений ИИ являлось основной идеей, лежащей в основе японского проекта разработки вычислительных систем пятого поколения, который осуществлялся в исследовательском центре ICOT. Европейское экономическое сообщество создало комиссию для организации исследований в области ИИ. В результате была сформирована программа Esprit. В Великобритании был образован комитет, возглавляемый Дж. Алви из фирмы British Telecom, и разработана своя национальная программа развития систем ИИ. В США работы в области ИИ были сосредоточены в программе СКИ (Стратегическая Компьютерная Инициатива), предложенной министерством обороны США.

Результатом этих исследований явилось создание новых, связанных с искусственным интеллектом, информационных технологий таких, как экспертные системы, нейронные сети, нечеткие системы, естественно-языковые системы.

Нефтегазовая отрасль является перспективным направлением внедрения систем искусственного интеллекта. В первую очередь это связано с наличием природного фактора, который порождает неопределенность и создает трудности при формализации процессов и явлений. Кроме того, технологические процессы (и соответствующие процессы обработки информации и управления), необходимые для преобразования нефти и газа из недр земли в конечный продукт, весьма разнообразны. Это - геологические и геофизические исследования, процессы бурения и разработки месторождений, транспорта и хранения нефти и газа, химические процессы и процессы переработки нефти и газа.

В этих условиях для современного развития систем управления в нефтегазовой отрасли становится актуальной подготовка инженерных кадров, владеющих методами и средствами искусственного интеллекта. Поэтому на кафедре АСУ для подготовки инженеров-системотехников открыта специализация: 220205 “Интеллектуальные автоматизированные системы”. Курсом “Системы искусственного интеллекта” начинается цикл дисциплин по указанной специализации.

Для успешного освоения данного курса необходимо, чтобы слушатели обладали хорошими знаниями в области моделирования, математической логики, автоматизированного управления. Необходимо также предварительное изучение таких дисциплин, как организация ЭВМ и систем, алгоритмические языки, технология программирования.

В результате изучения цикла дисциплин специализации планируется, что обучаемые будут знать и уметь использовать:

- основные модели, методы и инструментальные средства, предназначенные в АСОИУ для автоматизации интеллектуальных задач;

-описывать объекты, явления и процессы, связанные с конкретной областью специальной подготовки, использовать методы их научного исследования.

1. Искусственный интеллект как научное направление

Искусственный интеллект как научное направление: состояние и развитие. Экспертные системы, нейронные сети, нечеткие системы - основные направления развития и реализации ИИ.

1.1 Искусственный интеллект как научное направление: состояние и развитие

Информационные технологии активно и успешно проникают во все сферы деятельности человека. Искусственный интеллект является составной частью информатики, но при этом значительно расширяет ее возможности, позволяя решать плохо формализуемые задачи.

Искусственный интеллект (ИИ) - это программная система, имитирующая на компьютере мышление человека. Для создания такой системы необходимо изучать процесс мышления человека, решающего определенные задачи или принимающего решения в конкретной области, выделить основные шаги этого процесса и разработать программные средства, воспроизводящие их на компьютере. Следовательно, методы ИИ предполагают простой структурный подход к разработке сложных программных систем принятия решений.

Компьютерные программы, как правило, предназначены для решения строго определенных задач. Приспособить программу к решению новых задач можно, внеся в нее изменения, что требует тщательного просмотра программы и, соответственно, новых затрат времени. При внесении изменений в программе могут возникнуть дополнительные ошибки.

ИИ, как следует из самого названия, придает компьютеру черты разума. Методы ИИ упрощают объединение программ и дают возможность заложить в систему ИИ способность к самообучению и накоплению новой, полезной в дальнейшем информации. Человек может накапливать знания, не изменяя способ мышления и не забывая уже известных фактов. Аналогично работает система ИИ.

Искусственный интеллект опирается на знания о процессе человеческого мышления. Хотя точно неизвестно как работает человеческий мозг, и ученые продолжают постигать сложный механизм интеллекта, но имеющихся знаний для разработки программ ИИ вполне достаточно.

Основой человеческой деятельности является мышление. Целью называется конечный результат, на который направлены мыслительные процессы человека. Осуществление всех этих целей приводит к достижению главной цели. Каждый шаг на пути к главной цели имеет свою локальную цель. Мозг всегда сосредоточен на цели независимо от того, выполняет ли человек простую физическую работу или решает сложную интеллектуальную задачу. Цель заставляет человека думать. При проектировании систем ИИ всегда следует помнить о цели, для достижения которой они предназначены.

Основные проблемы ИИ - поиск и представление знаний, разработка теоретических представлений и создание для ЭВМ соответствующих программ наиболее общего характера.

Если предметом информатики является обработка информации, то к области ИИ следует относить только те случаи обработки информации, которые не могут быть выполнены с помощью простых, точных алгоритмических методов и когда число методов достаточно велико. Говорят так, что предметом изучения ИИ является любая интеллектуальная деятельность человека, подчиняющаяся заранее неизвестным законам. Иногда объектом исследования ИИ называют “все то, что еще не сделано в информатике”.

Информатика часто рассматривается как основа для моделирования и/или как инструмент для непосредственной проверки гипотез. Информатика и ИИ тесно связаны с лингвистикой, психологией и логикой. Все эти науки изучают явления, относящиеся к познанию, пониманию, умозаключению. В ИИ отражен интерес к мыслительным процессам.

ИИ это молодое научное направление, зародившееся в 60-е годы. Задачей этой науки является воссоздание на основе ЭВМ разумных рассуждений и действий. При этом имеют место две трудности: человек, принимая решение или выполняя то или иное действие, не знает точно алгоритм этого действия, например, как осуществляется понимание текста, доказательство теоремы, разработка плана действий и др.; вторая трудность заключается в том, что ЭВМ далека от человеческой компетентности.

Принято считать, что всякая задача, для которой неизвестен алгоритм решений, априорно относится к ИИ. Итак, сфера ИИ объединяет те области, в которых нет абсолютно точных методов решения задачи и которым присущи две характерные особенности:

- *использование информации в символьной форме* (буквы, слова, знаки, рисунки), что отличает от традиционного представления данных в числовой форме;

- *наличие выбора, т.е. недетерминизма*.

Недетерминизм отражает свободу действия, т.е. существенную составляющую интеллекта. Наличие недетерминизма носит фундаментальный характер, так как отражает реальную ситуацию выбора между многими вариантами в условиях неопределенности.

Наиболее известными направлениями развития ИИ являются:

- восприятие и распознавание образов;
- математика и автоматическое доказательство теорем;
- игры;
- решение повседневных задач (например, представленных в символьной форме и роботехника);
- понимание естественного языка.

Развитие ИИ определило и взаимосвязь многих научных дисциплин:

- психологии (в первую очередь в связи с тем человеческий мозг остается обязательным объектом исследования в ИИ);
- логики;
- лингвистики;
- биологии (модели передачи информации с помощью генов);
- информатики (самоорганизующиеся системы, поиск в базах данных, автоматическое программирование);
- медицины (помощь в диагностики);

- теории образовании и обучении.

1.2 Экспертные системы, нейронные сети, нечеткие системы - основные направления развития и реализации ИИ.

В ИИ выделяют следующие направления развития:

- экспертные системы (ЭС), или иногда их называют системы, основанные на знаниях (СОЗ);
- естественно-языковые системы (ЕЯ-системы);
- нейронные сети (НС);
- нечеткие системы (fuzzy-логика).

К этому можно добавить генетические алгоритмы и средства для извлечения знаний.

Интерес к системам ИИ возрастает из года в год. В 1995 общий объем продаж продуктов ИИ составил 1.1млрд. долл., из них 700 млн. долл. приходится на США.

Системы, основанные на знаниях, составляют более 70% от общего объема продаж. Средства ИИ разделяют на приложения и инструментальные системы (ИС). При этом доля ИС превышает более чем в десять раз долю приложений.

Экспертные системы - наиболее развитое направление ИИ. Представим общие сведения об экспертных системах.

ЭС - это информационная технология, использующая опыт и знания специалистов для решения плохо формализованных задач. ЭС представляет собой программный продукт, который решает задачи в интересующей нас предметной области. Появление ЭС и их использование в различных сферах человеческой деятельности явилось результатом исследований в области искусственного интеллекта.

Типовые задачи применения ЭС:

- интерпретация;
- прогноз;
- диагностика планирование;
- мониторинг;
- управление;
- обучение;
- ремонт.

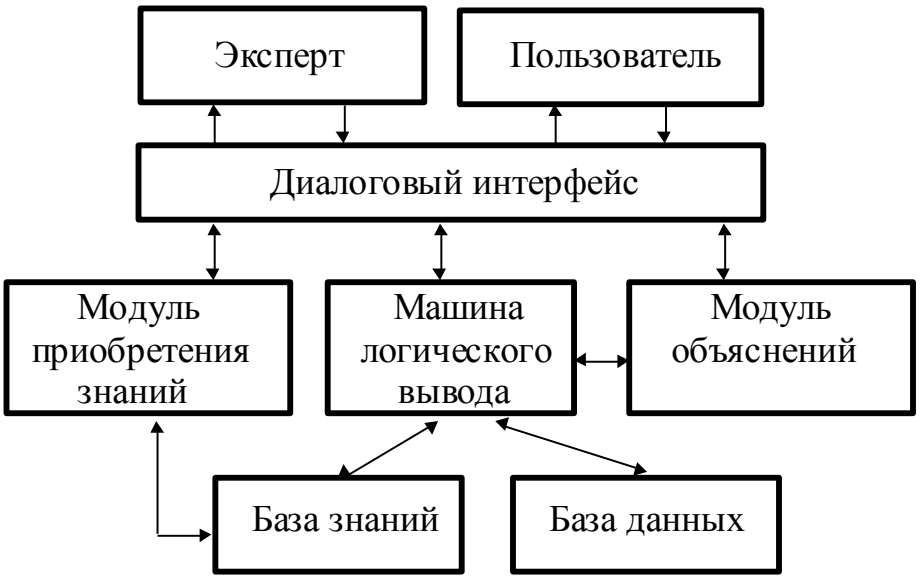


Рис.1 Структура ЭС

Этапы разработки:

- идентификация (формулирование задачи);
- концептуализация (определение места в системе, выбор инструментальных средств);
- формализация (формулировка правил);
- реализация;
- тестирование.

Выделяют два класса ЭС - статические системы и динамические (системы реального времени).

Основой для построения ЭС являются:

- оболочки (shell);
- инструментированные средства;
- языки ИИ;
- языки общего назначения.

Предпочтение отдается инструментальным средствам. При построении ЭС наибольшую популярность получили языки Prolog, Lisp и Си ++.

На рис.1 представлена обобщенная структура ЭС.

ЭС взаимодействует с пользователем или с экспертом через специальный диалоговый интерфейс, при разработке которого основной задачей является создание естественно-языковой среды общения. Эксперт через модуль приобретения знаний

имеет возможность дополнять систему новыми знаниями. Пользователь системы получает консультацию по введенному запросу, находясь в режиме активного диалога с машиной логического вывода через интерфейс пользователя. Машина логического достигает оценки целевого утверждения, взаимодействуя с базой знаний системы, используя при этом данные, заложенные в базу данных, или запрашивая их у пользователя. Модуль объяснений позволяет аргументировать пользователю обоснованность решения задачи и промежуточных выводов. Последнее качество делает систему более дружественной к пользователю, давая возможность показать логическую цепочку, которая привела ЭС к цели. Это свойство позволяет использовать ЭС не только для выдачи консультаций, но для обучения. В реальных системах набор модулей и их возможности могут быть намного шире.

Особое развитие в последние годы получили проблемно-ориентированные ИС. В области проблемно-ориентированных ИС ведущее место занимают фирмы Gensym (продукт G2), Talarian (Rtworks), Comdale Technologies (Comdale/C, Comdale/X).

В области работ по Е-Я системам выделяют следующие категории.

Е-Я-интерфейс к базам данных. Основная задача - преобразование Е-Я-запросов в SQL-запросы к БД. Лидирующее место в этой категории имеет фирма Symantec.

Е-Я-поиск в текстах и содержательное сканирование текстов (Natural Language Text Retieval and Contents Scanning Systems). В этой категории ИС осуществляют по запросам пользователя поиск, фильтрацию, сканирование текстовой информации. В отличие от предыдущей категории, где поиск осуществляется в БД, имеющей четкую и заранее известную структуру, в ИС этой категории поиск осуществляется в Е-Я-текстах, которые совершенно не структурированы. Существенный рост потребности в такого рода ИС связан с необходимостью обработки огромного количества текстов, доступных по сети Internet. Здесь можно отметить успехи фирм Excaliber Software, Architext Software, Verity, Ardilog Inc.

Масштабируемые средства для распознавания речи (Large Vocabulary Talkwtiter). ИС этой категории распознают голосовую информацию и преобразуют ее в последовательность символов. Ведущее место занимают фирмы Kurzweil Applied Informatics (ИС - VoiceMED), Dragon Systems Inc. (Dragon Wtiter), IBM (VoiceType).

Средства голосового ввода, управления и сбора данных (Voice Input and Control Products and Data Collection Systems). ИС четвертой категории в отличие от ИС

третьей категории ориентированы на работу со словарем не более 1000 слов и существенно ограничены в возможностях распознавания. В основном предназначены для ввода голосовых команд, управляющих работой некоторого продукта, например, программы сбора данных в тех приложениях, в которых у исполнителя заняты руки (одна из передовых фирм - Articulate Systems с ИС Voice Navigator).

Компоненты речевой обработки (Voice Recognition Programming Tools). ИС этой категории ориентированы для программистов, желающих добавить возможности по распознаванию речи в разрабатываемые ими приложения (фирмы AT&T, IBM).

Реально наибольшее распространение в этом направлении ИИ получили системы машинного перевода, программы контроля/исправления правописания и стиля, смешанные системы.

в) Нейронные сети (НС).

Технология НС нашла широкое применение в таких областях, как финансовая сфера (управление кредитными рисками, предсказание ситуаций на фондовом рынке, оценка стоимости недвижимости), химия (конструирование химических формул) распознавание оптических символов (optical character recognition-OCR), управление процессами.

Ведущие фирмы: California Scientific, Software Ward Systems Group.

2. Опыт и перспективы применения систем ИИ для решения задач автоматизированного управления в нефтегазовой отрасли

В настоящее время накоплен значительный опыт применения систем ИИ. В этом разделе рассмотрим только лишь некоторые примеры применения систем ИИ в нефтегазовой отрасли, в частности, экспертных систем.

ЭС TIGRESS (The Integrated Geoscience and Reservoir Engineering Software System) объединяет всевозможные средства системного анализа для всестороннего исследования месторождений. В систему включены три предметные области - геология, геофизика, нефтяное дело. ЭС LITHO помогает геологам интерпретировать данные каротажа нефтяных скважин. Эти данные включают кривые, отражающие измерения плотности, электрического сопротивления, звукопроводности и радиоактивности пород. Используются знания геологической обстановки района, чтобы охарактеризовать породы, через которые проходит скважина. DRILLING ADVISOR - ЭС, разработанная для помощи буровому мастеру при бурении нефтяных скважин,

реализует в интерактивном режиме советы, связанные с выявлением причин и возможными решениями при прихвате инструмента. Например, причинами прихвата инструмента могут быть коническая форма скважины, закупорка буровой колонны разбуренной породой, а мерами для преодоления этих трудностей могут быть подъем и/или спуск буровой колонны. ЭС MUD помогает инженерам обеспечить оптимальные свойства бурового раствора. Она делает это, диагностируя причины затруднений, связанных с применением раствора, и предлагает способы устранений. Возможные причины включают примеси, высокие температуры или давления и неправильное использование химических добавок. ЭС APDS (Automated Project Design System) предназначена для проектирования современных морских эксплуатационных установок для добычи нефти и газа; на основе оболочки системы Prospector реализована ЭС для обнаружения неисправностей в платформах, установленных в Северном море. ЭС CODA (Chemical Oil Dispersant Advisor) реализует советы по выбору и применению химических нефтяных дисперсантов.

Экспертная система выбора первоочередных объектов поисково-разведочных работ на нефть и газ, где явно проявляется образовательный характер ЭС. Определение очередности ввода обнаруженных сейсморазведкой ловушек нефти и газа в поисково-разведочное бурение является трудноформализуемой задачей, которая может быть решена путем последовательного анализа значений геологических факторов - тектонического, литологического и геохимического, определяющих условия генерации, миграции и аккумуляции углеводородов. Каждый из этих факторов оценивается большим числом признаков, поэтому в общем виде оценка перспективности ловушки является достаточно сложной и неоднозначно решаемой задачей. Наиболее распространенным путем ее решения является метод геологических аналогий, основанный на использовании опыта поисков и разведки в соседних, уже разбуренных зонах. При этом предполагается, что особенности строения и формирования залежей в пределах прогнозируемой зоны со значительной долей вероятности близки к характеристикам зоны с уже разбуренными структурами и разведанными скоплениями. Естественно, что наряду с месторождениями нефти и газа оказываются разбуренными и структуры, на которых по разным причинам получены отрицательные результаты. Обобщение опыта (экспертов) поисков и разведки соседних разбуренных зон с близкими или адекватными геологическими условиями и лежит в основе метода геологических аналогий.

Приведенные примеры использования ЭС в нефтегазовой отрасли относятся к статическим экспертным системам. В последние годы широкое развитие получили динамические ЭС, или экспертные системы реального времени (ЭСРВ), успех которых определен:

- переходом к специализированным (проблемно/предметным) инструментальным средствам, что влечет сокращение сроков разработки приложений и позволяет повторно использовать информационное и программное обеспечение (объекты, классы, правила, процедуры);

- использованием языков традиционного программирования и рабочих станций, что обеспечивает снижение требований приложений к быстродействию компьютера и объемам памяти, упрощение “интегрированности”, увеличение круга приложений;

- интегрированностью;

- открытостью и переносимостью;

- использованием архитектуры “клиент-сервер”.

ЭСРВ решают задачи, в которых:

- данные изменяются во времени, поступают от внешних источников и их необходимо хранить и анализировать;
- выполняются одновременно временные рассуждения о нескольких асинхронных задачах;
- обеспечивается механизм рассуждений при ограниченных ресурсах;
- обеспечивается “предсказуемость“ поведения системы, т. е. запущенная задача выполняется в строгом соответствии с временными ограничениями;
- моделируется окружающий мир, в котором отражаются различные состояния;
- протоколируются действия;
- эффективно и удобно обеспечивается наполнение базы знаний (эту возможность предоставляет объектно-ориентированный подход);
- обеспечивается настройка системы на решаемые задачи;
- реализуется пользовательский интерфейс для различных категорий.

Перспективным следует считать применение ЭС для поддержки диспетчерских решений, а также применение ЭС как реального средства системной интеграции при управлении сложными системами. Остановимся на этих положениях более подробно. На рис.2 показано, что основными объектами управления в АСУ

Рис.2 Объекты и системы управления

нефтегазовой отрасли являются технологические процессы и организационно-экономические системы. Часто последние в газовой отрасли называют системами производственно-хозяйственной деятельности (ПХД). Сюда входят бухгалтерский учет, материально-техническое снабжение, планирование, финансы, маркетинг и др.

На рис.3 представлены информационные системы отраслевой АСУ. Очевиден иерархический характер системы управления и интегрированность системы управления технологическими процессами и ПХД.

На рис.4 представлены тенденции развития АСУТП. Анализ показывает, что произошли качественные изменения в системе управления технологическими процессами, что существенно трансформировало функции и условия работы диспетчерского персонала. Диспетчер, работая в современной информационной среде, при возникновении нештатных ситуаций испытывает трудности в принятии решений. Это ситуации типа “что, если...”, в которых необходима подсказка-совет, основанный на правилах, сформированных на основе жизненного опыта и/или модельных расчетах. Такой советующей системой является ЭСРВ, что следует из рис.5**,** где представлены уровни современных человеко-машинных систем управления и соответствующая теоретическая база.

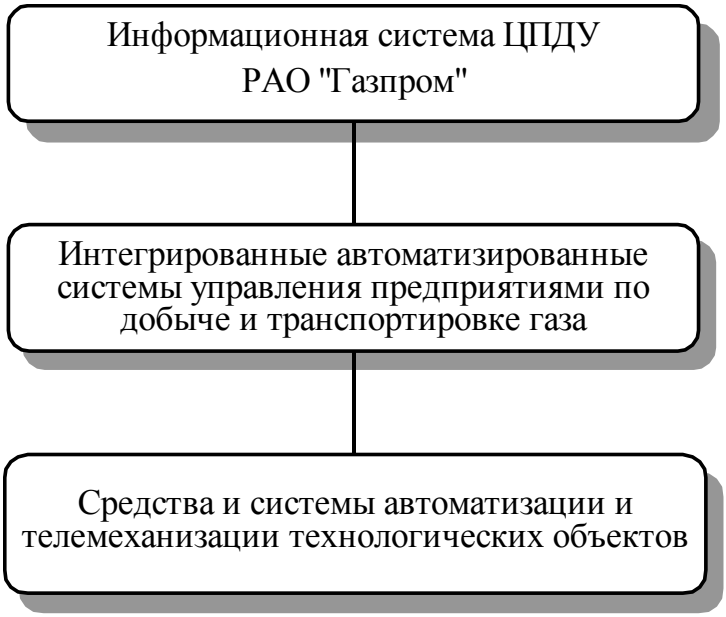


Рис.3 Информационные системы отраслевой АСУ

Рис. 4 Тенденции развития АСУ ТП

3. Классификация моделей представления знаний. Продукционные системы

Представление знаний и формальная логика. Представление знаний правилами и логический вывод. Граф "и/или" и поиск данных. Доска объявлений.

3.1 Представление знаний и формальная логика

Одной из наиболее важных проблем, которая возникает при построении систем основанных на знаниях, является проблема представления знаний. Знания предметной области могут быть записаны в виде правил и фактов. Структура механизма вывода зависит как от особенностей предметной области, так и от того, как знания организованы в ЭС. Различают следующие модели представления знаний:

- продукционные системы;
- семантические сети;

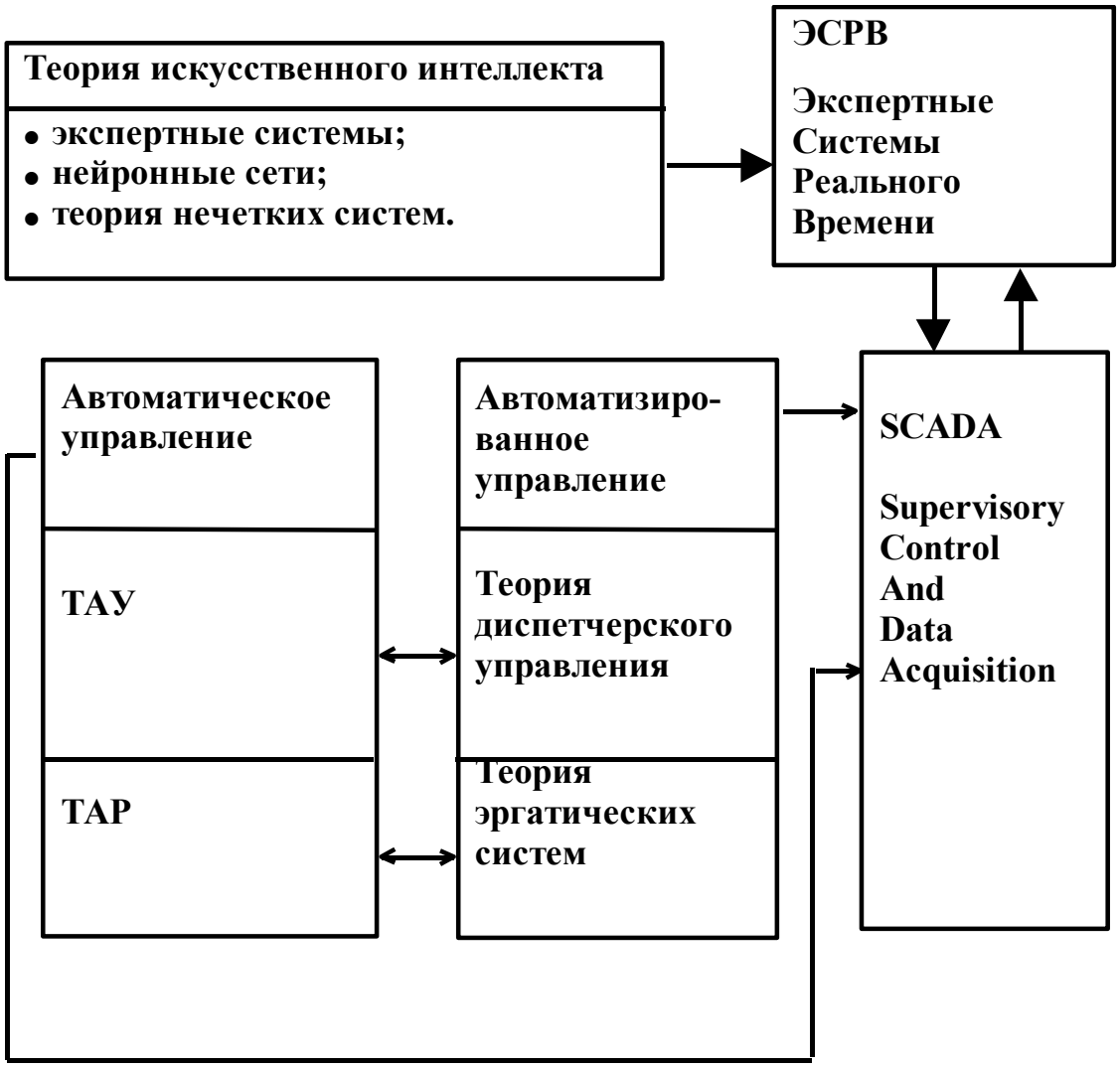


Рис.5 Уровни современных человеко-машинных систем управления

- фреймы;
- нечеткие модели;
- модели представления знаний на основе логики предикатов.

Важное место в ИИ занимает логический вывод, Формальная логика изучает методы правильных рассуждений, представляющих собой цепь умозаключений в логически последовательной форме. При этом рассуждения исследуются не с точки зрения смысла, а исходя из формы; отдельные элементы рассуждений могут замещаться какими-то другими, если это позволяют правила. Например, у Аристотеля, которому принадлежит известное рассуждение о Сократе (“Сократ-человек”, “Все люди - смертны”, “Сократ - смертен”), замещающие символы носят название переменных. Например, Сократ может быть заменен другим именем, и, в конце концов, обобщением (“Все люди”).

Но в обычном языке недостаточно только переменных; возникает естественно необходимость в словах: “если, тогда, и, или, принадлежит, влечет и др.” Эти слова имеют фундаментальное значение в рассуждениях и поэтому такие слова называются “ операторы”.

Для понимания формальных систем следует обратиться к истории, сравнению силлогистики и математической логики.

Силлогизм - форма дедуктивного умозаключения, в которой из двух высказываний (посылок) субъектно-предикатной структуры следует новое высказывание (заключение) той же логической структуры. Таким образом, силлогистика (теория дедуктивного вывода), разработанная Аристотелем (“Органон”) и явилась первой логической теорией дедуктивного рассуждения и служила отправной точкой для разработки формальной логики. Вплоть до 17 века силлогистика считалась совершенной в своей законченности логической теорией и составляла традиционно логический элемент любого гуманитарного образования. Математическая логика выработала более общую, чем силлогистика логическую систему - исчисление предикатов и строгие методы логического анализа.

Здесь уместно вспомнить еще один термин - умозаключение. ***Умозаключение*** - умственное действие, связывающее в ряд посылок и следствий мысли различного содержания. Умозаключение реализует в плане “внутренней речи” присущие индивидуальному (или общественному) сознанию нормы и типы такой связи, которые и являются в каждом отдельном случае психологической основой умозаключения.

Если эти нормы и типы совпадают с правилами и законами логики, то умозаключение по своему результату равносильно логическому выводу, хотя в общем случае умозаключение и логический вывод качественно различны.

Одна из задач формальной логики - выявление неоднозначностей и изучение отдельных этапов рассуждений или выводов, когда шаг за шагом строго доказывается их правильность вне зависимости от используемой интерпретации, и не остаются неясности ни на одном этапе. Формальная логика на начальном этапе рассматривалась как составная часть философии, приобретая со временем математический характер. Лишь начиная с трудов Г.Фреге и Г.Пеано и с монументального труда Б.Рассела и А Уайтхеда" Основая математики" (1913г.) формальная наука полностью отделилась от разговорного языка и выделилась в математическую дисциплину.

Формальная система представляет собой совокупность чисто абстрактных объектов (не связанных с внешним миром), в котором представлены правила оперирования множеством символов в чисто синтаксической трактовке без учета смыслового содержания (или семантики).

Формальная система *определена*, если:

1. Задан конечный алфавит, т.е. конечное множество символов. Если алфавит заранее предполагается конечным, то иногда он называется также словарем. В этом случае в нем различают константы, переменные, операторы.

2. Определена процедура построения формул (слов) формальной системы. Это определяет конкретную синтаксическую конструкцию формул. Иногда ее также называют грамматикой формул, которые представляют собой правильно построенные последовательности символов.

3. Выделено некоторое множество формул, называемых аксиомами.

4. Задано конечное множество правил вывода, которые позволяют получать из некоторого конечного множества формул другое множество формул.

3.2 Представление знаний правилами и логический вывод. Граф и/или и поиск данных. Доска объявлений

Одной из важных задач, которую инженер знаний должен разрешить до работ, связанных непосредственно с реализацией ЭС на ЭВМ, является выбор модели представления знаний оптимальной для конкретной задачи.

Моделью в случае представления знаний правилами является система продукций.

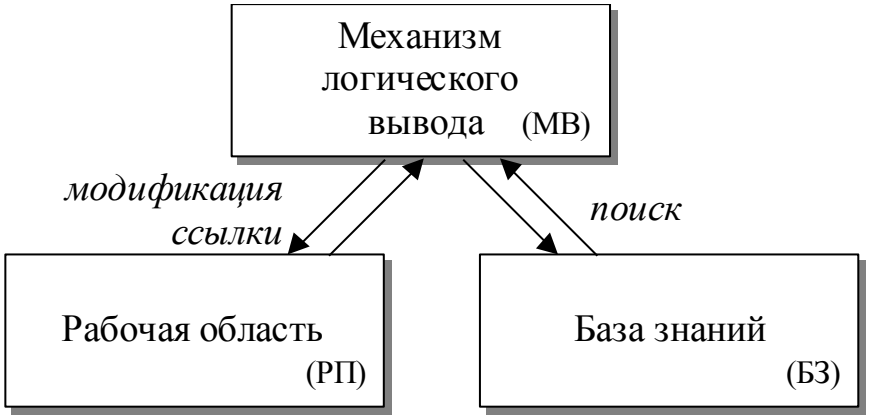


Рис.6 Продукционная система

Продукционные правила, составляющие базу знаний (БЗ), есть выражения вида:

Е С Л И (условие),
Т О (действие).

Это операция *"импликация*", или как ее иначе называют "логическое следование".

Если факты, входящие в условную часть правила, принимают значение "истина", то выполняется действие A $\rightarrow$ B. Если высказывание типа "если A, то B" истинно, то из истинности A (true) необходимо следует истинность B. Но в случае, когда A - ложно (false), нельзя судить об истинности B. Поэтому для случая, если A - ложно и при любых значениях B, высказывание истинно.

A	B	A $\rightarrow$ B	
1	1	1	T
1	0	0	F
0	1	1	T
0	0	1	T

Продукционная система состоит из трех компонентов (рис.6):

База знаний (БЗ) содержит набор правил. Рабочая память (РП) предназначена для кратковременного хранения предпосылок и результатов вывода. Механизм логического вывода (МВ) использует правила в соответствии с содержимым рабочей памяти.

В свою очередь механизм вывода условно можно представить как

$$MB = < S, R, V, W >,$$

где V - операция выборки из РП и из БЗ подмножества активных данных и правил, используемых в очередном цикле работы механизмов вывода; S - операция сопоставления, определяющая множество означиваний, т.е. множество пар “ правило - данные”; R - процесс разрешения конфликтов, при этом определяется какое из означиваний будет выполняться первым; W - процесс выполнения означиваний.

Продукционную БЗ можно представить в виде графа И/ИЛИ. Так как правила состоят из условной и заключительной частей, а условная часть связана либо логической функцией И, либо логической функцией ИЛИ, то любое правило можно представить графом, где на нижнем уровнем расположены узлы-условия, а на верхнем уровне - узел-заключение. При этом связь верхнего и нижнего уровней осуществляется с помощью логической функции И либо ИЛИ.

Механизм логического вывода - это стратегия поиска решения на графе логического вывода. Обычно это сводится к механизму выбора соответствующего правила из конфликтного набора. ***Конфликтный набор*** - это множество правил, которые могут быть применены на конкретном этапе логического вывода. Управление логическим выводом имеет большое значение, так как существенен порядок выдачи запросов пользователю и оптимальность набора запрашиваемых данных.

Существует два способа использования правил: ***прямая цепочка рассуждений; обратная цепочка рассуждений***.

Рассмотрим пример прямой цепочки рассуждений (рис. 7).

Пусть в базе данных или рабочей области (БД или РП) имеются факты F, R, D, M, B, а в базе знаний - три правила.

Первым выполняется только верхнее правило, согласованное с данными. В нашем примере на первом шаге может быть выполнено только третье правило. Заметим также, что правило выполняется только один раз. Вывод реализуется в виде цикла “сопоставить-выполнить”, причем в каждом цикле выполненная часть выбранного правила обновляет рабочую память. В результате содержимое рабочей области преобразуется от первоначального наполнения к целевому.

Итак, при прямой цепочке рассуждений на каждом этапе логического вывода анализируется текущее состояние рабочей области, содержащей факты, и активизируются те продукции, условная часть которых принимает значение “истина”.

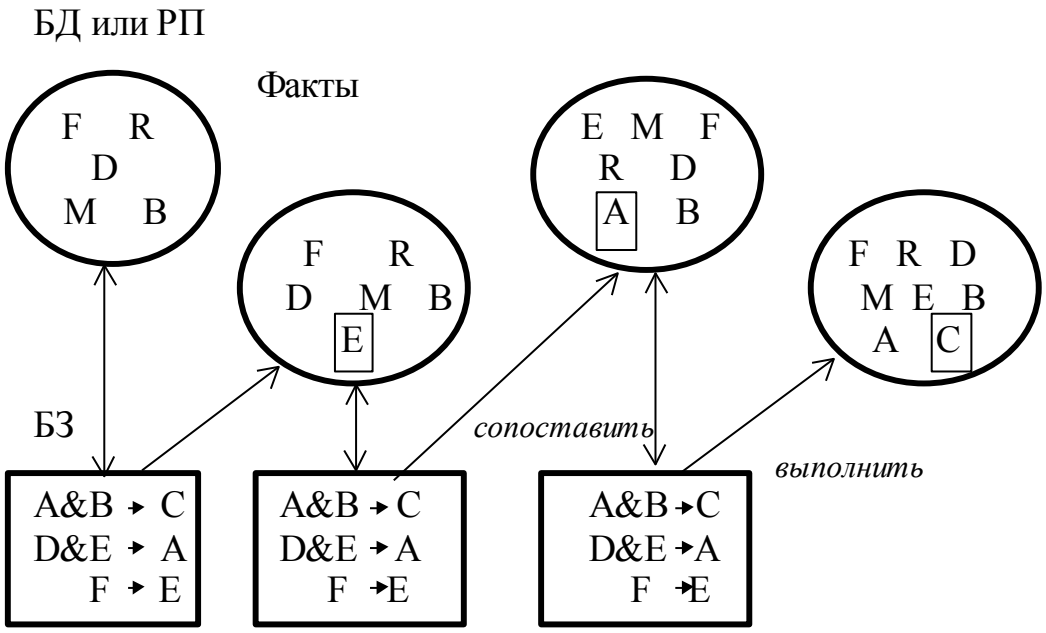


Рис.7

Выбранное правило активизируется и факт, являющийся заключением, заносится в рабочую область. Процесс продолжается до определения целевого утверждения.

Преимущество прямой цепочки рассуждений - простота реализации. Недостаток - в избыточности активизированных правил, приводящих к формально справедливым заключениям, но не имеющим никакого отношения к реальному целевому заключению. Эта проблема усугубляется с ростом числа правил. Получается, что при прямой цепочке рассуждений генерируется конфликтный набор, который может быть решен двумя способами: либо ограничением на генерацию конфликтного набора, либо введением специального алгоритма.

В этом случае заслуживает внимания обратная цепочка рассуждений. При этом методе сначала активизируются правила, способные подтвердить целевую гипотезу. Затем факты, входящие в условную часть этих правил, устанавливаются промежуточными целями, и система старается их означить. При этом нет необходимости осуществлять полный перебор правил БЗ.

Пример обратной цепочки рассуждений (рис.8).

Цель - существует ли ситуация С. Цель ищем в правилах и фактах.

Шаг1: Так как найденное правило A&B приводит к цели С, то система решает, что она должна установить факты А и В, чтобы вывести С.

Шаг2: Установление факта A требует выполнения правила D&E. Для этого необходимо установить факты D и E. Ищется факт D.

Шаг3: Система находит D и переходит к поиску E.

Шаг4: В соответствии с найденным правилом, связывающим F и E, ищется F.

Шаг5: Нахождение F в БД.

Шаг6: Выполнение третьего правила F → E и занесение E в РП.

Шаг7: Выполнение второго правила БЗ и нахождение A; занесение A в РП.

Шаг8: Выполнение первого правила в БЗ и определение C.

Обратная цепочка более распространена, чем прямая цепочка. Системе обратного вывода соответствуют многие задачи, например, задачи диагностики, возникающие при отказе оборудования, заболеваниях.

Примером такой системы может служить одна из первых экспертных систем - ЭС MYCIN. ЭС MYCIN помогает врачам выбирать антимикробную терапию для

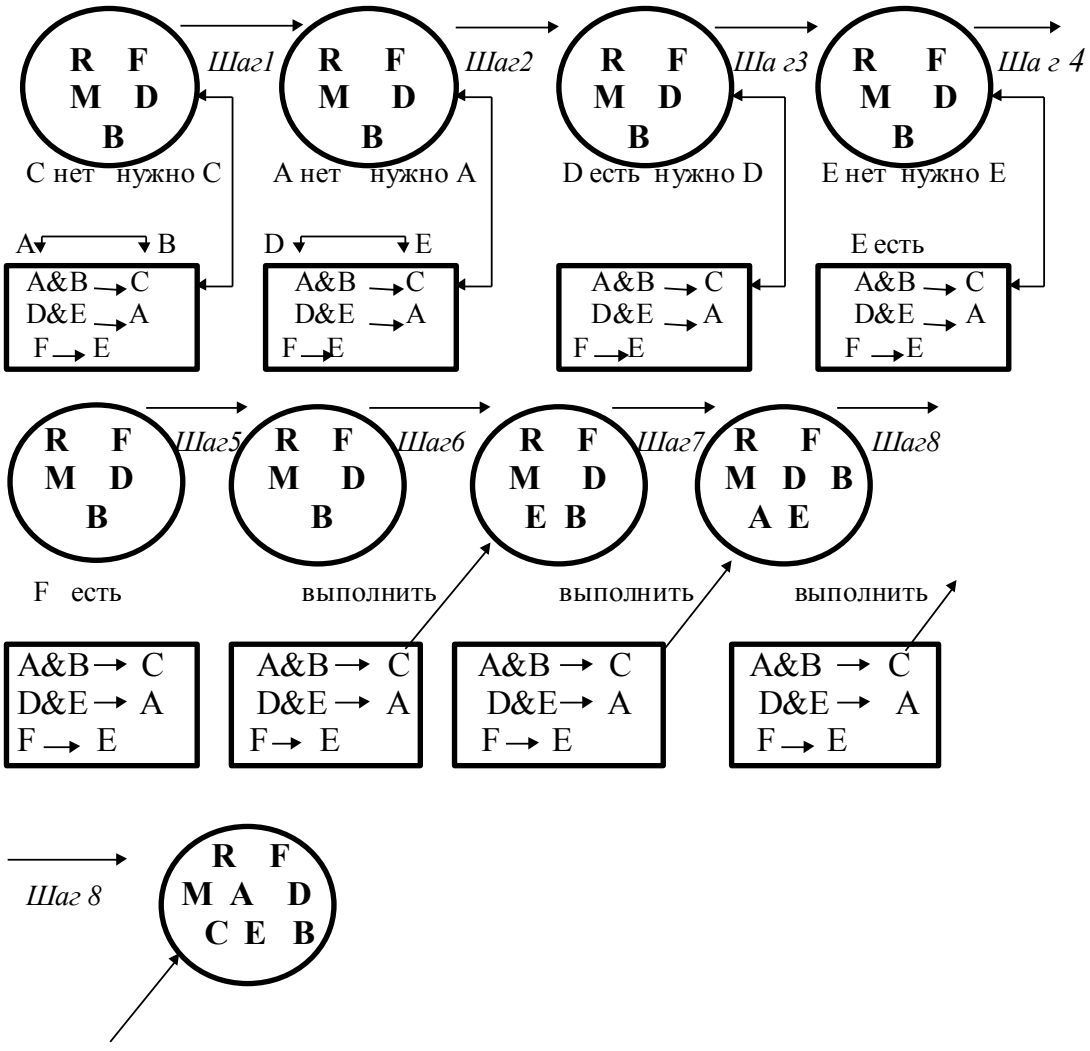


Рис.8

госпитализированных больных. Система определяет причину инфекции и рекомендует лекарственное лечение. То есть осуществляет диагноз и рекомендует лечение инфекционных заболеваний. Система реализована на языке LISP и имеет обратную цепочку рассуждений, содержит механизмы вычисления вероятностей и построение объяснений в процессе рассуждения. Факты запоминаются в виде триплета "*объект-атрибут-значение*". Триплет характеризуется фактором достоверности CF, значение которого лежит в диапазоне от –1 (отрицание) до +1 (утверждение). Знания эксперта кодируются в виде набора продукций.

Как видно системы продукций характеризуют простота создания и понимания отдельных правил, модификации знаний, простота механизма логического вывода. Слабыми сторонами продукционных систем является неясность взаимных отношений правил, сложность оценки целостного образа знаний, низкая эффективность обработки. При решении небольшой задачи выявляются только сильные стороны системы продукций. В случае увеличения объема знаний и необходимости решать более сложные задачи требуется структурирование знаний. В этом случае рекомендуется "доска объявлений".

Впервые "доска объявлений" была предложена в системе Hearsay-II. На "доске объявлений" иерархически определены гипотезы, на самом верхнем уровне находятся заключения, на самом нижнем - факты, на промежуточных - промежуточные выводы (гипотезы). Вокруг "доски объявлений" сгруппированы правила продукций, которые называют источниками знаний. Они действуют на соответствующих уровнях гипотез или между уровнями. "Доска объявлений" исполняет роль канала связи между источниками знаний. Восходящий вывод называется выводом, управляемым данными, а нисходящий- выводом, управляемым моделью. Это соответствует прямой и обратной цепочкам рассуждений. Подобная схема предпочтительнее с точки зрения систематизации знаний и эффективности вывода.

4. Модели представления знаний (семантические сети, фреймы, нечеткие системы)

4.1 Представление знаний семантической сетью

Обращение к семантической сети для представления знаний связано с известной идеей о том, что "память" формируется через ассоциации между понятиями.

“ Ассоциативная память” как одна из теорий, объясняющих феномен памяти, была известна еще в древние времена. В информатику эта идея вошла в связи с работами по использованию простых ассоциаций для представления смысла слов в базе данных.

Базовый функциональный элемент семантической сети - это структура из двух компонент: узла и дуги. Узел - это объект, концепция, событие, понятие. Дуга - отношение между парами понятий. Каждое отношение отражает факт. Дуги имеют направленность и возможно отношение “*субъект - объект*”. Так как каждый из узлов может быть соединен с любым числом других узлов, то можно получить сеть фактов. С точки зрения логики предикатов базовая структура семантической сети интерпретируется так: предикат - это дуга, аргументы - узлы.

Семантические сети наглядны. Их популярность определена также рядом сочетаний “связь-отношение”, такими как **is_a** (является), **has-part** (имеет часть), которые позволяют строить иерархию понятий. В этом случае устанавливается свойство иерархии наследования в сети, когда элементы нижнего уровня наследуют свойства элементов более высокого уровня.

Особенностью осуществления логического вывода при использовании представления знаний семантической сетью является его неотделимость от базы знаний. Обычно вывод осуществляется процедурами, сопоставляющими сеть базы знаний с подсетью, представляющей собой целевое утверждение или вопрос, при этом основным требованием является непротиворечивость представленных знаний.

4.2 Представление знаний фреймами

Идея представления знаний фреймами принадлежит М.Минскому. Эту модель называют психологической моделью памяти человека и его сознания. Фреймы относятся к психологическим понятиям, связанными со способами восприятия информации. С помощью фреймов осуществляется концептуальное моделирование. В основе теории фреймов лежит восприятие фактов посредством сопоставления, полученной извне информации, с конкретными элементами и значениями, а также с определенными в нашей памяти рамками, отражающими каждый концептуальный объект. Считают, что теория фреймов в большей степени относится к теории постановки задач, а не к методам решения.

В книге «A Framework for Representing Knowledge in the Phychology of Computer Vision» М.Минский писал: «...Суть теории. Когда человек оказывается в новой

ситуации, он вызывает в своей памяти основную структуру, именуемую фреймом. ***Frame*** (рамка) - это единица представления знаний, заполненная в прошлом, детали которой могут быть изменены согласно текущей ситуации. Каждый фрейм можно рассматривать как сеть, состоящую из нескольких вершин и отношений. На самом верхнем уровне фрейма представлена фиксированная информация: факт, касающийся состояния объекта, который обычно считается истинным. На последующих уровнях расположено множество так называемых терминальных слотов (slot-щель), которые должны быть заполнены конкретными значениями и данными. В каждом слоте задается условие, которое должно выполняться при установлении соответствия между значениями. Слот либо сам устанавливает соответствие, либо это делает более мелкая составляющая фрейма».

Рассмотрим некоторые из основных понятий теории фреймов.

1. ***Базовый тип***. В виде базовых типов запоминаются наиболее важные объекты. Если изменяется точка зрения на этот объект, то меняются лишь значения, но структура остается неизменной, так как базовый тип определяет сущность объекта, с которым работаем.
2. ***Процесс сопоставления.*** Это процесс, в ходе которого проверяется правильность выбора фрейма. Это процесс осуществляется в соответствии с текущей целью и информацией, содержащейся во фрейме. Фрейм содержит условия, ограничивающие значения слота, а цель используется для определения, какое из этих условий, имея отношение к данной ситуации, является нужным в данное время.
3. ***Иерархическая структура.*** Так как фрейм объединяет всю информацию об объекте, то для структуризации этой информации необходима иерархическая структура. Кроме того, она позволяет реализовать функцию наследования.
4. ***Межфреймовые сети.*** Предоставляют возможность объединения фреймов между собой.
5. ***Значение по умолчанию.*** Существует предположение о том, что решения о занесении в долговременную память не принимаются, пока не распределены терминальные значения. А до этого момента во фрейме хранятся значения по умолчанию. Выводы, которые делаются на основании значений по умолчанию, называются выводами по умолчанию.
6. ***Отношение “абстрактное - конкретное”, “целое - часть”.*** Отношение “абстрактное-конкретное” характерно тем, что на верхних уровнях располагаются

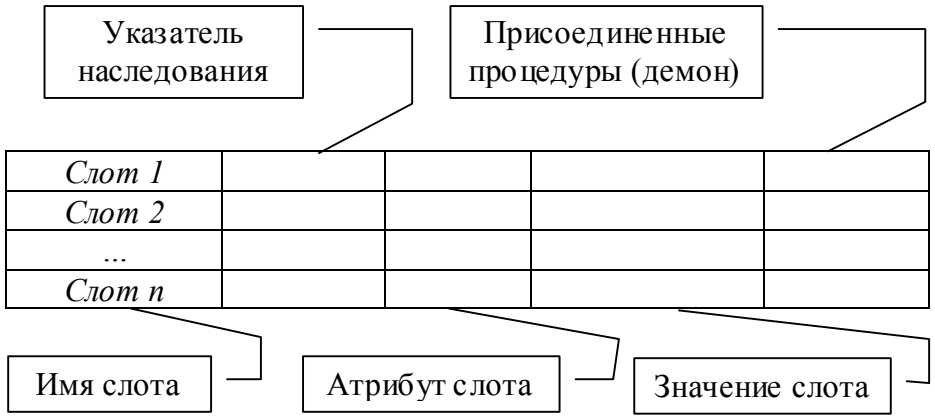


Рис.9. Структура данных фрейма

абстрактные объекты, на нижних - конкретные. Причем имеет место свойство наследования.

Достоинства фреймовой системы в том, что те элементы, которые присутствуют в описании объектов или событий, группируются и обрабатываются как единое целое. Во фреймах объединяются как декларативные, так и процедурные знания. Фреймовая система - иерархическая структура, узлам которой являются подобные фреймы.

Рассмотрим структуру данных фреймов (рис. 9).

Имя фрейма - идентификатор, являющийся уникальным для каждого фрейма в системе. Имя слота - идентификатор, присваиваемый слоту фрейма. Он должен быть уникальным для каждого фрейма. Обычно имя слота не несет смысловой нагрузки, но есть специальные ключевые имена: IS-A - показывающий фрейм - родитель данного фрейма, DDESENDANTS - показывающий прямой дочерний фрейм, FINEDRY - пользователь, определяющий фрейм, DEFINEDON - дата определения фрейма и т.д. Эти слоты называются системными и используются при редактировании базы знаний и управления логическим выводом. Указатель наследования, показывающий, какую информацию об атрибутах слотов фрейма верхнего уровня наследуют слоты с тем же именем производных фреймов. Обычно употребляют указатели наследования:

- U - unique (уникальный). Показывает, что каждый фрейм может иметь слоты с различными значениями;
- S - same (такой же). Все слоты имеют одинаковые значения;
- R - range (установление границ). Значения фреймов нижнего уровня должны находиться в пределах границ, указанных в слотах верхнего уровня;

- O - override (игнорировать). При отсутствии указания значения слота фрейма верхнего уровня становится значением слота нижнего уровня.

Атрибут слота указывает тип данных слота. Это FRAME (указатель), INTEGER (целый), REAL (действительный), BOOL (логический), LISP (присоединенная процедура), TEXT (текст), LIST (список), TABLE (таблица) и другие. Значение слота - это значение, соответствующее указанному для данного слота атрибуту. Демон - процедура автоматически запускаемая при выполнении некоторого условия. Наиболее часто используемые – это: IF - NEEDED (если нужно), IF - ADDED (если добавлено), IF - REMOVED (если удалено).

В языке представления знаний фреймами отсутствует специальный механизм управления выводом. Поэтому пользователю для вывода предоставляются присоединенные процедуры. Язык представления знаний фреймами считают языком высокого уровня универсальности, что позволяет пользователю писать свои программы управления выводом с помощью присоединенных процедур. Во фреймовых системах используются три способа управления выводом: с помощью присоединенных процедур, с помощью служебной процедуры и, используя механизм наследования.

4.3 Представление нечетких знаний

Нечеткая логика, расплывчатые множества, нечеткие системы Заде - это небольшой список терминов, относящихся к одному тому же, т.е. то к системам, в которых трудно определить границу принадлежности. Знания не всегда могут быть описаны точно. Термин “ нечеткость” весьма неоднозначен и поэтому в ИИ методы обработки нечеткости подразделяют на: недетерминированность выводов, многозначность, ненадежность, неполноту, нечеткость или неточность.

5. Модель представления знаний на основе логики предикатов

Понятие предиката. Элементы языка предикатов. Представление знаний с помощью логики предикатов

5.1 Понятие предиката

Логика означает систематический метод рассуждений. Наиболее распространены две логические системы: базисная (исчисление высказываний) и исчисление предикатов (как система, имеющая большие возможности). Исчисление

трактуют как совокупность правил оперирования с какими-то символами. ***Исчисление высказываний*** - совокупность правил, используемых для определения истинности или ложности некоторой комбинации высказываний. Это - *двузначная* логика, в этой логике работает только “да” и “нет”. Древние называли это принцип, как “закон исключенного третьего”. Начальный смысл логики предикатов - объяснение логических основ естественного языка.

В связи с тем, что предикаты с одной стороны - элементы языковой системы, а с другой стороны - это логическая система, рассмотрим предикат с обеих позиций.

Латинское *predicatum* означает сказуемое. В узком смысле это логическое сказуемое, указывающее на свойство отдельного предмета. В широком смысле предикат это отношение, сложное сказуемое. Часто логику предикатов называют языковой системой, которая оперирует предложениями на естественном языке, но в пределах заданных синтаксических правил.

В логике ***предикат*** – понятие, определяющее предмет суждения и раскрывающее его содержание. Предикат аналогичен понятию функции и раскрывает отношение между двумя объектами. Но в отличие от числовой функции аргументы предиката не обязательно должны принимать числовое значение, а значениями самих функций для предиката являются высказывания.

С точки зрения модели представления знаний и механизма логического вывода достоинством логики предикатов является хорошо формализованный математический аппарат. В связи с этим упрощается процесс программирования механизма логического вывода.

Систематизированная трактовка логики предикатов проводится выбором из естественного языка тех его составных частей, которые не содержат внутри себя нечеткостей. Сокращенный естественный язык, используемый для вывода логических рассуждений, не должен содержать ни синонимов, ни омонимов.

5.2 Элементы языка предикатов. Представление знаний с помощью логики предикатов

Основными элементами языка логики предикатов служат: переменные, константы, предикаты, термы, кванторы. Исходными в исчислении высказываний являются атомы. Из них строятся формулы. Но в исчислении высказываний отсутствуют переменные, и это затрудняет описание предметной области. Логика предикатов (первого порядка) является расширением языка высказываний, так как

основным объектом здесь является переменное высказывание, истинность или ложность которого зависят от значения его аргументов. Решение задачи означает ее описание с помощью формул логики предикатов первого порядка, а затем доказательство общезначимости или противоречивости полученной конечной формулы.

Логика предикатов первого порядка работает с переменными и представляет собой, в некотором смысле, аналог теории функций.

Итак, для описания предметной области в логике предикатов используются:

- *константы;*
- *переменные;*
- *термы;*
- *предикаты;*
- *кванторы.*

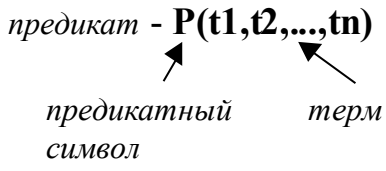
Если ***константа*** - это символ, обозначающий индивидуальный объект или понятие, то ***переменная*** - символ, используемый в разное время для обозначения различных объектов, при этом для определения области действия переменных используются ***кванторы.***

Термы рекурсивно определяются следующим образом:

1. Константа есть терм.
2. Переменная есть терм.
3. Пусть f - n-мерный функциональный символ, то составной терм f(t1, ..., fn) - это тоже терм.
4. Никаких других термов, кроме тех, которые определены по позициям 1-3, нет.

Предикатный символ P используется для представления отношений между объектами данной предметной области. Предикатные символы используются для построения высказываний, и это отличает предикатный символ от функционального, который используется для построения термов.

Предикат состоит из предикатного символа и соответствующего ему упорядоченного множества термов, являющихся его аргументами:



Предикат как логические функции принимает только два значения: истина и ложь.

Определение предиката. Если P есть n-местный предикатный символ языка, t1, t2, ..., tn - термы, то P(t1, t2, ...,tn) называется атомарной или элементарной формулой языка, а сама запись означает, что истинно высказывание гласящее, что объекты t1, t2, ...,tn связаны отношением P.

Из атомарных формул с помощью логических связок (*и, или, не, следует* и др.) и кванторов строятся сложные формулы языка.

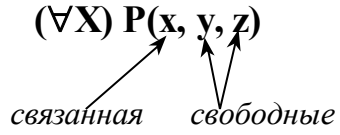
Квантор - логическая операция, дающая количественную характеристику предметной области. Различают два вида кванторов:

∀ - квантор всеобщности, и ∃ - квантор единственности.

Если записываем выражение ∀X P(x), то область истинности предиката совпадает с областью значения переменной x, или для любого x выражение P(x) истинно.

Если записываем выражение ∃X P(x), то это означает, что существует x, при котором выражение P(x) истинно.

Связанными переменными называют переменные, непосредственно стоящие после знака квантора. Различают и *свободные* переменные, то есть те, на которые знак квантора не распространяется.



Когда все переменные предиката являются связанными, то такой предикат называется предложением. Предикаты, которые используются в представлении знаний, обычно являются предложениями.

Пример 5.1 Записать с помощью формул I-го порядка следующие выражения.

Для лечения любой давно известной болезни в медицине имеется лекарство. Появились новые болезни, для лечения которых не существует лекарств.

A(x) - x - является давно известной болезнью.

B(x) - для лечения болезни x существует лекарство.

~B(x) - против болезни x не существует лекарств.

(∀x) A(x) - для всех болезней, которые являются давно известными.

(∃x) ~A(x) - существует болезни недавно известные (новые) - появились новые болезни.

((∀x)A(x) ->B(x)) - для всех давно известных болезней существует лекарство.

((∃x)~A(x) -> ~B(x)) - новые болезни, для которых нет лекарств.

((∀x) A(x) -> B(x))∧((∃x)~A(x) -> B(x)) - показывает, что есть объекты, отношения - атомарные формулы.

Предикаты служат для описания языковой системы, для которой необходимо задать следующие компоненты.

1. Множество знаков (т.е. алфавит).
2. Полное определение слов через знаковые последовательности (морфемы); морфемы - наименьшая часть слова, которая является той минимальной единицей языка, за которой закреплено то или иное содержание; морфема неделима на более простые единицы, обладающие тем же свойством.
3. Грамматические правила образования предложений и слов, то есть синтаксис или синтаксические правила.
4. Неотъемлемой частью языка предикатов является задание двух типов слов, первый тип описывает сущности изучаемой предметной области и называется термы; второй тип описывает атрибуты и поведение атрибутов, то есть отношения и называется предикат.

Определение логической формулы в логике предикатов представляют собой правила построения сложных предложений из простых предложений и синтаксических единиц, включающих описание отношений между основными понятиями атомарных предикатов. При этом формулы логики предикатов строятся как символьная система безотносительно к смыслу или понятиям описываемого мира.

Между понятиями описываемого мира и предикатными формулами имеются определенные соответствия:

	Описываемый мир	**Предикатная система**
1.	Сущности *Установление соответствия*	Константы *Константы принимаются за имена*
2.	Функциональные отношения	Формулы
3.	Концептуальные отношения	Атомарные формулы

В зависимости от выполнения или невыполнения концептуальных отношений, которые описываются логическими формулами, задаются значения “истина” и ”ложь”. Подобным образом устанавливается определенное соотношение между формулами

логики предикатов и концептами реального мира. Далее работает механизм логического вывода, т. е. система рассуждений и полученный результат осмысливается в терминах реального мира. Таким образом, семантика в систему логики предикатов, изначально определенную независимо от описываемого мира, вносится через интерпретацию результатов.

6. Метод резолюций

Интерпретация. Метод резолюций для логики высказываний. Метод резолюций для логики предикатов. Хорновские дизъюнкты

6.1 Интерпретация

Интерпретация формул в логике высказываний - это приписывание входящим в формулу атомам значений: “истина” и “ложь”. Интерпретация в логике предикатов - это задание области значений переменных и присвоение значений всем константам, предикатным и функциональным символам, встречающимся в этой формуле, в соответствии с рядом правил.

Формулы логики предикатов первого порядка определяются рекурсивно в соответствии со следующими правилами.

1. Атом - есть формула.

2. Если A и B формулы, то ~A, $A \wedge B$, $A \vee B$, $A \rightarrow B$ есть тоже формулы.

3. Если A(x) формула, то $(\exists x)$ A(x), $(\forall x)$ A(x) - формулы.

4. Никаких других формул, кроме тех, которые определены по правилам 1-3 нет.

Интерпретация в логике предикатов приводит в итоге к определению характера логических формул, которые могут быть общезначимыми или противоречивыми или выполнимыми.

Формула является ***общезначимой*** тогда и только тогда, когда не существует никакой интерпретации, при которой формула принимает значение ложь:

$$A \vee \sim A.$$

Формула является ***противоречивой***, тогда и только тогда, когда не существует ни какой интерпретации, при которой формула принимает значение истина:

$$A \wedge \sim A.$$

Формула является ***выполнимой***, тогда и только тогда, когда существует хотя бы одна интерпретация, что формула принимает значение - истина.

Практически для исчисления высказываний интерпретация сводится, в конечном счете, к построению таблицы истинности, на основании которой можно судить об общезначимости или противоречивости. Интерпретация для логики предикатов становится сложной проблемой вследствие размерности задачи. Процесс интерпретации осуществляется с формулами, представленными в некотором каноническом виде. Если для логических формул в исчислении высказываний такой канонической формой служат КНФ или ДНФ, то в логике предикатов это - *префиксная нормальная форма*.

По определению формула находится в префиксной нормальной форме тогда и только тогда, когда формула состоит из двух элементов: *префикса*, состоящего из цепочки кванторов, и *матрицы*, представляющей собой *бесквантор*ную *формулу*.

Обозначим через Q любой квантор, а через M матрицу. Тогда префиксная форма может быть выражена как (Qx1) (Qx2)... (Qxn) M.

Для записи в префиксной нормальной форме используются следующие эквивалентные формы:

1. $A \leftrightarrow B = (A \rightarrow B) \wedge (B \rightarrow A)$
2. $A \rightarrow B = \sim A \vee B$
3. $A \vee B = B \vee A$ или $A \wedge B = B \wedge A$
4. $(A \vee B) \vee C = A \vee (B \vee C)$ или $(A \wedge B) \wedge C = A \wedge (B \wedge C)$
5. $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$ или $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
6. $A \vee F = A$ или $A \wedge T = A$
7. $A \vee T = T$ или $A \wedge F = F$
8. $A \vee \sim A = T$ или $A \wedge \sim A = F$
9. $\sim(\sim A) = A$
10. $\sim(A \vee B) = \sim A \wedge \sim B$ или $\sim(A \wedge B) = \sim A \vee \sim B$
11. $(Qx)A(x) \vee B = (Qx)(A(x) \vee B)$
12. $(Qx)A(x) \wedge B = (Qx)(A(x) \wedge B)$
13. $\sim((\forall x)A(x)) = (\exists x)(\sim A(x))$
14. $\sim((\exists x)A(x)) = (\forall x)(\sim A(x))$
15. $(\forall x)A(x) \wedge (\forall x)B(x) = (\forall x)(A(x) \wedge B(x))$
16. $(\exists x)A(x) \vee (\exists x)B(x) = (\exists x)(A(x) \vee B(x))$
17. $(Q1x)A(x) \vee (Q2x)B(x) = (Q1x)(Q2y)(A(x) \vee B(y))$

18. $(Q3x)A(x) \vee (Q4x)B(x) = (Q3x)(Q4y)(A(x) \wedge B(y))$

Если первые десять формул непосредственно связаны с исчислением высказываний, то формулы с 11 по 18 относятся уже к логике предикатов.

6.2 Метод резолюций для исчисления высказаваний

Основная идея решения задач с помощью логики состоит в:

- описании задачи с помощью логических формул;
- доказательстве общезначимости или противоречивости всей совокупности этих формул.

Так как в логике предикатов имеется очень большое число интерпретаций, то невозможно доказать истинность для всех возможных интерпретаций. Поэтому в логике предикатов нет алгоритмов, доказывающих или проверяющих общезначимость. Но имеются алгоритмы, подтверждающие общезначимость. В основе этих алгоритмов положено следующее утверждение: “формула общезначима тогда и только тогда, когда ее отрицание противоречиво”. Поэтому большинство автоматических алгоритмов поиска доказательств состоят в установлении противоречивости отрицания данной формулы.

Наиболее распространенным методом доказательства общезначимости является ***метод резолюций***, который представляет собой процедуру поиска опровержения отрицания исходной гипотезы.

Пусть S - множество дизъюнктов, которые представляют стандартную форму формул F. F- противоречива тогда, когда противоречиво S. Если S содержит пустой дизъюнкт, то S невыполнимо. Если S не содержит пустой дизъюнкт, то проверяется, может ли он быть получен из S.

Дизъюнктом называется дизъюнкция литер или литералов. Множество литер - это синоним дизъюнкта. Дизъюнкт, содержащий n литер - “n-мерный дизъюнкт”. Дизъюнкт, не содержащий ни одной литеры, называется пустым. А пустой дизъюнкт всегда ложен, потому что в нем нет литер, которые могли бы быть истинными при любых интерпретациях. Пусть КНФ распадается на: $A1 \vee A2 \vee ... \vee Ak$, тогда A1...Ak - дизъюнкты.

Итак, метод резолюций сводится к поиску пустого дизъюнкта.

Литеры A и ~A называют ***контрарная пара*** {A, ~A}.

Пусть в дизъюнкте A1 существуют литера l_1 контрарная литере l_2 в дизъюнкте A2. Вычеркнем l_1, l_2 из указанных дизъюнктов и построим дизъюнкцию оставшихся, полученный дизъюнкт называется ***резольвентой***, и он представляет собой логическое следствие исходных. Получение из множества S пустого дизъюнкта является доказательством невыполнимости S.

Пример 6.1

1. Получить резольвенту из исходных дизъюнктов

F1: $A \vee B$,

F2: $\sim A \vee C$.

Резольвента R: $B \vee C$

2. Доказать, что формула C является логическим следствием:

F1: $A \rightarrow B$

F2: $B \rightarrow C$

F3: A

Решение.

Добавим F4: ~C.

Перепишем исходные формулы, используя выше приведенные эквивалентные формы:

F1: $\sim A \vee B$.

F2: $\sim B \vee C$.

F3: A;

F4: ~C.

Вычеркивание контрарных пар из F2 и F4 приводит к резольвенте R1: ~B.

Сравнение R1: ~B и F1: $\sim A \vee B$ дает R2: ~A и далее R3: (пустой дизъюнкт).

Результат. Формула C является логическим следствием данных формул.

6.3 Метод резолюций для логики предикатов

Как было показано, если дизъюнкты не содержат переменных, то нахождение в двух дизъюнктах контрарных пар является простой процедурой. Если дизъюнкты содержат переменные, то их необходимо ***унифицировать***. Это означает: найти такую подстановку, чтобы исходные дизъюнкты содержали контрарные пары. Здесь термин “унификация” означает сопоставление. Унификацию используют для того, чтобы показать как литеры могут быть приведены в соответствие между собой. Любую подстановку можно представить с помощью множества упорядоченных пар ti/xi

$$S = \{t1/x1, t2/x2, ..., ti/xi\}$$

Здесь, терм ti заменяет переменную xi.

В логике предикатов для приведения к стандартной форме записи необходимо реализовать ряд этапов.

1.Исключение символов импликации. Например,

$$(\forall x)(A(x) \rightarrow B(x)) = (\forall x)(\sim A(x) \vee B(x))$$

2.Перенос символа отрицания внутрь формулы, то есть каждое отрицание должно относится к одной атомарной формуле. Это удачно реализуется с использованием формул де Моргана.

3.Разделение переменных. Переменная, связанная некоторым квантором, в пределах области действия этого квантора, может быть заменена любой другой переменной, не встречающейся ранее в данной формуле. Это нужно, чтобы каждому квантору соответствовала только одна, свойственная ему переменная.

$$(\forall x)(A(x))\wedge(\exists x)B(x)$$

заменяется на:

$$(\forall x)(A(x))\wedge(\exists y)B(y)$$

4.Исключение квантора существования из формул. Переменная, связанная квантором существования, может быть заменена функцией, называемой *сколемовской* функцией. Например, $\exists x\ A(x)$ можно заменить $A(b)$ при $x = b$ и $b = const$

5.Преобразование к префиксной нормальной форме. Все кванторы общности переводятся в начало формулы. При этом область действия каждого квантора включает всю формулу. Все переменные связаны, и формула состоит из префикса и матрицы. Тогда префикс можно просто убрать или опустить. Например,

$$(\forall x)\ (\text{человек}(x) \rightarrow \text{мать}(x, y)).$$

Смысл не изменится, если $(\forall x)$ убрать

$$\text{человек}\ (x) \rightarrow \text{мать}\ (x, y)$$

6. Приведение к конъюнктивной нормальной форме. На этом этапе формулу записывают в виде совокупности дизъюнктов, соединенных между собой функцией конъюнкции.

7. Исключение символов конъюнкции.

Каждая формула исчисления предикатов эквивалентна совокупности дизъюнктов, состоящих из литералов и соединенных между собой символами дизъюнкции. Слово совокупность подчеркивает, что порядок дизъюнктов не играет роли.

Если множество исходных гипотез непротиворечиво, то надо добавить к нему дизъюнкты с отрицанием целевого высказывания. Получение пустого дизъюнкта показывает, что доказываемое высказывание истинно. Сравнение дизъюнктов начинается всегда с целевого дизъюнкта.

Так как механизмом логического вывода при представлении модели знаний на основе логики предикатов служит метод резолюций, и доказательство общезначимости сводится к противоречивости отрицания исходной гипотезы, то необходимо ввести понятие “хорновский дизъюнкт”.

6.4 Хорновские дизъюнкты

В дизъюнкте одни литеры содержат, а другие не содержат знак отрицания. ***Хорновский дизъюнкт*** - это дизъюнкт, в котором есть только одна позитивная литера.

$$P,Q,R \rightarrow S \qquad S\text{:-}P,Q,R.$$

Если соблюдаются условия, то S - истинно.

$$P \wedge Q \wedge R \rightarrow S$$
$$\sim(P \wedge Q \wedge R) \vee S$$
$$\bar{P} \vee \bar{Q} \vee \bar{R} \vee S$$

Имеется два вида хорновских дизъюнктов:

- с заголовком (A:-B,C,D);
- без заголовков (:-B,C).

Пустой дизъюнкт не имеет заголовка. Из двух хорновских дизъюнктов с заголовками всегда получаем хорновский дизъюнкт с заголовком. Поэтому, чтобы вывести пустой дизъюнкт необходимо, по крайней мере, хотя бы один хорновский дизъюнкт без заголовка - это и есть целевой дизъюнкт.

Хорновские дизъюнкты являются основой программных систем для автоматического доказательства теорем и, в частности, языка Пролог.

7. Основы языка Пролог

7.1 Логическое программирование

Свое название Пролог получил от слов “ПРОграммирование в терминах ЛОГики” (PROgramming in LOGic).

Теоретической основой Пролога стали идеи логического программирования, предложенные Робертом Ковальским (Эдинбург) в начале семидесятых. Основная идея логического программирования состоит в следующем: на языке математической логики предикатов первого порядка, точнее на некотором ограниченном подмножестве этого языка – хорновских дизъюнктах, задача записывается как набор

высказываний, отношений между высказываниями и правил вывода одних высказываний из других. Затем с помощью формальных процедур, основанных на доказательстве теорем, проверяется справедливость исходных высказываний.

Логическая программа состоит из утверждений. Каждое утверждение выражает некоторые характеристики объектов или отношения между объектами, например «Яблоко красное» или «Имеет место событие A, если имеют место события B, D, C». Последнее утверждение может быть записано с помощью хорновских дизъюнктов в виде `A:-B,C,D`, или в логике предикатов первого порядка в виде формулы $B \wedge C \wedge D \rightarrow A$ или $\neg B \vee \neg C \vee \neg D \rightarrow A$. Именно набор хорновских дизъюнктов с заголовками составляет логическую программу, например:

```
A:-B,C,D.
B:-E,F.
C:-G.
D.
T.
F.
G.
```

Описание предметной области является статическим, оно только определяет базу данных, в которой хранится информация об объектах и отношениях между ними.

Логическая программа начинает выполняться только после ввода предполагаемого заключения, которое называется запросом или вопросом к программе и представляет собой условие или конъюнкцию нескольких условий. Например:

```
?- A.       имеет ли место событие A
?- C.       имеет ли место событие C
?- A,C,D.   имеют ли место одновременно события A,C и D.
```

Входящие в вопрос предикаты могут иметь аргументы, например

```
?- A(X).    найти такие X, что имеет место событие A(X)
```

В логической программе получение (вычисление) ответа на вопрос означает доказательство существования рассматриваемого объекта. Процесс завершается успешно, если вопрос к программе логически следует из исходных условий. Для выполнения программы используется встроенная система автоматического поиска вывода или автоматического доказательства теорем. Одна из главных проблем логического программирования – это поиск эффективных методов организации выполнения логических программ.

Ковальский и ван Эмден предложили два альтернативных подхода к прочтению текстов логических программ – ***процедурный*** и ***декларативный***. Правило A:-B,C,D. можно трактовать двояко, *во-первых*, как логическое утверждение о том, что высказывание A истинно, если одновременно истинны утверждения B, C и D, и, *во-вторых*, как определение процедуры A, для выполнения которой надо выполнить действия B, C и D.

Первый интерпретатор языка Пролог был разработан группой Алэна Колмероэ из Марсельского университета в 1972 г. Программа была реализована на Фортране и работала крайне медленно. В последующие годы появилось большое количество реализаций интерпретаторов и компиляторов языка Пролог для различных операционных систем, отличающихся синтаксисом, некоторыми особенностями исполнения и удобством программирования.

7.2 Терминология

Любое предложения в Прологе называется ***предикатом***. Единственная структура данных ***терм*** – это объект данных: константа, переменная, структура, или составной терм.

Константами являются поименованные конкретные объекты или конкретные отношения. Существует два вида констант - атомы и целые числа. ***Атом*** - последовательность букв, цифр и знака подчеркивания, начинающаяся со *строчной* буквы. Значение константы - это ее имя, так константа 2 может только соответствовать числу 2, а константа abrakadabra может соответствовать только символам abrakadabra. Символьная константа должна начинаться со строчной буквы или заключается в кавычки.

Переменные обозначают единственный, но *неопределенный объект*. Символы переменных должны начинаться с *прописной* буквы или символа подчеркивания. Понятие переменной в Прологе отличается от принятого в процедурных языках программирования, она не рассматривается как выделенный участок памяти. Переменную можно считать локальным именем для некоторого объекта. Переменная ***конкретизирована***, если имеется объект, который обозначает эта переменная. Переменная, состоящая только из символа подчеркивания «_», называется ***анонимной*** и используется в том случае, если имя и значение переменной несущественно. Областью действия переменной является утверждение (факт, правило или вопрос), ее

содержащее. В пределах утверждения одно и то же имя принадлежит одной и той же переменной. Но каждая анонимная переменная отличается от всех других анонимных переменных одного и того же утверждения.

Структура (составной терм) - объект, состоящий из совокупности других объектов. Структура строится из ***функтора*** (имени), представляющего собой атом, и последовательности термов, называемых аргументами. Число аргументов показывает ***арность*** структуры. Обозначение: *"функтор/арность"*. Константы имеют нулевую арность. Структуры, имеющие одинаковое имя, но различную арность, различны. Аргументы заключаются в круглые скобки и разделяются запятыми, функтор записывается перед открывающей круглой скобкой, например,

```
мужчина (авраам).                  мужчина/1
отец (авраам, исаак).              отец/2
дата (сентябрь, 1, 1998)           дата/3
имеет(X, книга(Название, Автор)).  имеет/2
                                   книга/2
```

Структуры арности 1 и 2 могут быть записаны в операторной форме, если атом, используемый как главный функтор в структуре, объявить ***оператором***. Оператор имеет следующие свойства: *позиция, приоритет, ассоциативность*. Позиция указывает место оператора по отношению к аргументам. Приоритет указывает порядок выполнения действий. Каждый оператор связан со своим классом приоритета. Класс приоритета представляет собой целое число, оператор с большим приоритетом имеет класс, близкий к 1. Свойство ассоциативности указывает на порядок выполнения операторов одного приоритета. Левоассоциативный оператор должен иметь слева операции одинакового или низшего приоритета, а справа - операции низшего приоритета. Например, все арифметические операции являются левоассоциативными. В Прологе определены арифметические операторы и операторы сравнения. Пользователь может определить свой оператор, обратившись к встроенному предикату "*op/3*":

```
op(P, S, N).
```

где **P** – приоритет оператора**,** **S** – спецификатор, задающий позицию и ассоциативность, например

fx определяет постфиксный оператор

xf префиксный оператор

xfx инфиксный оператор

xfy	инфиксный правоассоциативный оператор
yfx	инфиксный левоассоциативный оператор

N – имя оператора, должно быть атомом.

Оператор «+» объявлен как `op(500, yfx, +).`

Программа на Прологе есть совокупность утверждений. Утверждения хранятся в базе данных Пролога, и база данных Пролога может рассматриваться как программа на Прологе. В конце утверждения ставится **«.»**. Существует два типа утверждений:

Факт – это утверждение о существовании некоторого отношения между аргументами, обозначаемого именем предиката. Факты всегда истинны.

Правило можно рассматривать как факт, значение истинности которого зависит от значений условий, образующих *тело* правила. Заголовок правила имеет точно такую же форму, как и факт. После заголовка стоит обозначение **«:-»** (которое читается как *«если»*), а затем располагается тело правила. Каждое условие, входящее в тело, называется *подцелью*. Для того чтобы заголовок правила оказался истинным, необходимо, чтобы каждая подцель была истинной для связанных друг с другом отношением ***«и»*** (обозначается **«,»**), или одна из подцелей была истинной для связанных отношением ***«или»*** (обозначается **«;»**). В действительности, правила, в которых встречается символ **«;»,** преобразуются Прологом в два правила, имеющих только конъюнкции подцелей.

После записи утверждений в базу данных вычисления могут быть инициированы вводом ***запроса***, или ***целевого утверждения***. Запрос выглядит так же, как и подцели, образуется и обрабатывается по тем же правилам, но не входит в базу данных. Обозначается **«?-»** с арностью 1. Обычно запрос записывается в операторной форме: за знаком **«?-»** следует ряд утверждений в виде конъюнкций.

Пример 7.1.

Программа содержит набор фактов "`путешествие/3`", каждый из которых устанавливает, что можно доехать из одного города (первый аргумент) в другой (второй аргумент) на некотором виде транспорта (третий аргумент), и правило "`можно_путешествовать/2`", позволяющее установить прямую или косвенную связь между двумя городами, через третий – промежуточный – город:

```
путешествие(нью_йорк, бостон, поезд).
путешествие(нью_йорк, принстон, поезд).
```

```
путешествие(нью_йорк, вашингтон, поезд).
путешествие(нью_йорк, вашингтон, самолет).
путешествие(бостон, портленд, автобус).
путешествие(бостон, портленд, поезд).

можно_путешествовать(A, B):- путешествие(A,B,_);
                             путешествие(A,C,_),
                             путешествие(C,B,_)).
```

В соответствии с приведенными данными правило "можно_путешествовать" даст положительный результат для городов Нью-Йорк и Портленд, так как можно добраться из Нью-Йорка в Бостон поездом, а из Бостона в Портленд на автобусе. Варианты запросов:

```
?- можно_путешествовать(нью_йорк, портленд).
?- можно_путешествовать(X,Y).
?- можно_путешествовать(нью_йорк, X).
?- можно_путешествовать(X, портленд).
```

Пример 7.2

Определение оператора для преобразования температуры по Цельсию в температуру по Фаренгейту. Задача решается в два этапа. На первом нужно определить предикат (правило), который занимается собственно преобразованием. Пусть это будет «to/2»:

```
to(C,F):- F is (C*9/5+32).
```

На втором – объявить предикат «to/2» как инфиксный левоассоциативный оператор с приоритетом 100:

```
op(100, yfx, to).
```

Для использования в программе можно записать «Cel to Far», причем переменная Cel должна быть конкретизированной, а Far неозначенной переменной.

Основу вычислительной модели логических программ составляет алгоритм ***унификации*** (*отождествления, сопоставления с эталоном*). Унификация является основой автоматической дедукции и логического вывода в задачах искусственного интеллекта.

Пролог пытается отождествить термы при доказательстве, или согласовании, целевого утверждения. Например, в примере 1 для согласования запроса

```
?- можно_путешествовать(нью_йорк,X).
```

целевое утверждение `путешествие(нью_йорк, X,_)` отождествляется с фактом

```
путешествие(нью_йорк, бостон, поезд).
```

в результате чего переменная X конкретизируется значением `бостон (X = бостон)`.

Переменные, входящие в утверждение, отождествляются особым образом – ***сопоставляются***. Факт доказывается для всех значений переменных. Правило доказывается для всех значений переменных в головном целевом утверждении при

условии, что хвостовые целевые утверждения доказаны. Переменные принимают конкретные значения на время доказательства целевого утверждения.

Терм X сопоставляется с термом Y по следующим правилам. Если X и Y – константы, то они сопоставимы, только если они одинаковы. Если X является константой или структурой, а Y – неконкретизированной переменной, то X и Y сопоставимы и Y принимает значение X (и наоборот). Если X и Y - структуры, то они сопоставимы тогда и только тогда, когда у них одни и те же главный функтор и арность, а также каждая пара их соответствующих аргументов сопоставима. Если X и Y – неконкретизированные переменные, то они сопоставимы, в этом случае говорят, что они ***сцеплены***. Например,

Терм1	**Терм2**	**Результат унификации**
путешествие (нью_йорк,бостон, поезд)	путешествие(нью_йорк, принстон, поезд)	нет
путешествие(нью_йорк, X,_)	путешествие(нью_йорк, бостон, поезд)	да: X = бостон
путешествие(X, Y, Z)	T	да:T=путешествие(X,Y,Z)
путешествие(X, X, Z)	путешествие(X, Y, Z)	нет
3 + 2	5	нет
X	Y	да: X = Y

Пролог всегда находит *наиболее общий унификатор*. Так в последнем примере для X и Y существует бесконечное множество унификаторов: X = 1, Y = 1; X = 2, Y = 2; X = *a*, Y = *a*…, но Пролог находит наиболее общий X = Y.

В процессе унификации Пролог выбирает первое из тех утверждений программы, голова которого сопоставима с целевым утверждением. Если удастся согласовать тело утверждения, то целевое утверждение согласовано, иначе Пролог переходит к следующему утверждению, голова которого сопоставима с целевым утверждением… При согласовании тела утверждения, представленного списком подцелей, в случае если некоторая из подцелей не может быть согласована (и она не является первой подцелью), Пролог *возвращается назад*, чтобы повторно проанализировать предыдущую подцель запроса. Если подцель – первая в запросе, неудача сопоставления приводит к неудаче всего запроса. При возврате назад ликвидируются все конкретизации переменных, выполненные последним запросом. Возврат может также быть осуществлен, если необходимо получить несколько

вариантов решения задачи. Метод повторного согласования целевого утверждения называется ***механизмом возврата***.

То есть Пролог – строго последовательный язык: выбор утверждений осуществляется слева направо строго в том порядке, в котором они записаны в программе.

7.3 Арифметика в Прологе

Пролог не предназначен для программирования задач с большим количеством арифметических операций, для этого используются процедурные языки программирования, но в любую Пролог-систему включают все обычные арифметические операторы (+, -, *, /, div, mod) и операторы сравнения.

Вычисление арифметических выражений производится при помощи встроенного системного предиката **is**, который определен как инфиксный оператор, левый аргумент которого число или неконкретизированная переменная, а правый – арифметическое выражение. Предикат **is** не является встроенным решателем уравнений.

Попытка доказательства целевого утверждения `X is Y` заканчивается успехом в одном из следующих случаев:

– `X` – неконкретизированная переменная, а `Y` – число;

– `X` – число, которое равно результату вычисления `Y`.

Цель `X is Y` не может быть согласована вновь.

7.4 Рекурсия

Рекурсия – это алгоритмический метод, используемый в Прологе для достижения такого же эффекта, какой реализуется при употреблении итеративных управляющих конструкций в процедурных языках (например, циклы for или while). Правила являются *рекурсивными*, если в качестве одного из условий содержат обращение к самим себе.

В процессе выполнения программы рекурсивное правило вызывает само себя до тех пор, пока не будет выполнено условие окончания рекурсии, т.е. не произойдет обращение к другому утверждению данного определения, не содержащему рекурсивного вызова.

На каждом уровне рекурсии переменные трактуются как новые. Это

происходит потому, что, рекурсивно обращаясь к тому же правилу, Пролог-система действует так, как будто имеет дело каждый раз с новым правилом (повторно кодирует имена переменных).

Для примера 7.1 можно записать рекурсивное правило "`можно_путешествовать/2`", показывающее возможность попасть из одного города в другой через любую последовательность промежуточных пунктов.

```
можно_путешествовать(A, B):- путешествие(A,B,_).
можно_путешествовать(A, B):- путешествие(A,C,_),
                             можно_путешествовать(C,B).
```

7.5 Списки и бинарные деревья

Рекурсивными могут быть не только правила, но и структуры данных. Тип данных является рекурсивным, если он допускает структуры, содержащие такие же структуры, как и они сами. Важным рекурсивным типом данных является ***список***, структура арности 2, второй аргумент которой является тоже списком.

Списки широко используются для представления деревьев синтаксического разбора, карт городов, математических объектов (графов, формул, функций) и т.д.

В Прологе предусмотрена скобочная форма записи, при которой элементы списка заключены в квадратные скобки и разделены запятой: [a, b, c, d] - список, содержащий атомы a, b, c, d. Первый элемент называют *"головой"* списка, все последующие - это *"хвост"* списка. Имеется также специальная форма для представления списка с "головой" X и "хвостом" Y, где для разделения X и Y используется вертикальная черта. Например, [X | Y] , список [a, b, c, d] можно записать как [a | [b,c,d]], допустима и такая запись [a, b | [c,d]].

[ ] - пустой список.

Пример 7.3

1. Определение принадлежности элемента к списку.

Предикат “элемент/2” состоит из двух утверждений. Первое – это факт: “элемент X является элементом некоторого списка, если он совпадает с головным элементом этого списка”. Второе – рекурсивное правило: “элемент X является элементом некоторого списка, если элемент X является элементом списка, полученного из данного исключением головного элемента”.

```
элемент(X, [X | L]).
элемент(X, [Y | L]):- элемент(X, L).
```

2. Слияние двух списков в третий.

Первый аргумент – исходный список, второй - добавляемый список, третий – результирующий список.

```
слияние([], L, L]).
слияние([H | T], L, [H | NewList]):- слияние(T, L, NewList).
```

Если заданы любые два списка, то программа позволяет сформировать третий. Если заданы все три списка, программа проверяет, является ли третий список результатом слияния первых двух. Например, для выделения первого списка при известных других [c,d] и [a,b,c,d] можно сделать такой запрос:

```
?- слияние(X, [c,d], [a,b,c,d]).
```

3. Исключение из списка элемента.

Первый аргумент – исходный список, второй – исключаемый элемент, третий – новый список.

```
удаление([H | L], H, L).
удаление(([H | L], X, [H | NewList]):- удаление(L, X, NewList).
```

4. Длина списка.

```
длина([],0).
длина(([H | L], N):- длина(L, N1),
                     N is N1 + 1.
```

Для списков с большим количеством элементов обработка не всегда бывает эффективной. Например, L – список, описывающий множество их первых 1000 натуральных чисел, тогда при ответе на запрос

```
?- элемент(3333,L).
```

Прологу придется проверить все 1000 чисел, прежде чем заключить, что такого числа нет.

Представление множества ***бинарным деревом*** позволяет добиться лучшего результата. Бинарное дерево определяется рекурсивно как имеющее *левое поддерево, корень* и *правое поддерево*. Левое и правое поддеревья сами являются бинарными деревьями.

Например, дерево, представленное на рис.10, можно представить следующим термом: бд(бд((бд(nil, d, nil), b, бд(nil, e, nil))), a, бд(nil, c, nil)). Атом nil используется для представления пустого дерева.

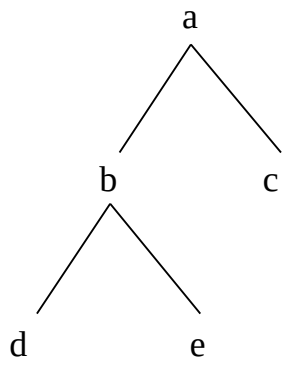


Рис. 10 Бинарное дерево.

Для эффективного использования бинарное дерево должно быть *упорядочено* таким образом, чтобы любой элемент в левом поддереве был меньше, чем значение корня, а любой элемент в правом поддереве – больше (рис. 11).

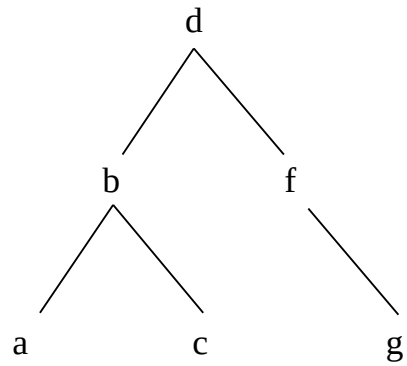


Рис. 11 Упорядоченное бинарное дерево.

Пример 7.4

1. Добавление элемента в упорядоченное бинарное дерево.

При добавлении если элемент меньше корневого элемента, то он добавляется к левому поддереву, если же больше, то – к правому. Для проверки порядка следования использованы операторы `@<` и `@>`.

```
вкл_бд(nil, X, бд (nil, X, nil)).
вкл_бд(бд(Лд,K,Пд), X, бд(Лд1,K,Пд)):- X @< K,
                                       вкл_бд (Лд,X,Лд1).
вкл_бд(бд(Лд,K,Пд), X, бд(Лд,K,Пд1)):- X @> K,
                                       вкл_бд (Пд,X,Пд1).
```

2. Построение упорядоченного дерева из списка (в общем случае, неупорядоченного).

```
список_в_дерево([], nil).
список_в_дерево([H | L], Бд):- список_в_дерево(L, Бд1),
                               вкл_бд(H, Бд1, Бд).
```

3. Построение отсортированного списка из упорядоченного дерева.

Отсортированный список для упорядоченного бинарного дерева `бд(Лд,K,Пд)`, где `Лд` имеет отсортированный список СЛ, а Пд – отсортированный список СП, получается

слиянием списков СЛ и [К | СП].

```
дерево_в_список(nil,[]).
дерево_в_список (бд(Лд,К,Пд),С):- дерево_в_список(Лд,СЛ),
                                  дерево_в_список(Пд,СП),
                                  слияние(СЛ,[К|СП],С).
```

7.6 Управление ходом выполнения программы

Встроенный механизм поиска с возвратом может привести к поиску ненужных решений, в результате чего теряется эффективность, например, когда желательно найти только единственное решение. В других случаях необходимо продолжать поиск дополнительных решений, даже если целевое утверждение уже согласовано. В подобных ситуациях необходимо управлять процессом поиска с возвратом.

Пролог обеспечивает два инструментальных средства, которые дают возможность управлять механизмом поиска с возвратом: предикаты ***fail*** - используется для поддержания поиска с возвратом, и ***cut***, или *отсечение* (обозначается «**!**»), - используется для предотвращения поиска с возвратом.

В определенных ситуациях бывает необходимо выполнить поиск с возвратом, чтобы найти другие решения. Встроенный предикат ***fail*** вызывает состояние неудачи при доказательстве целевого утверждения и, следовательно, инициирует поиск с возвратом. Действие предиката *fail* равносильно эффекту от сравнения 2 = 3 или другой невозможной подцели.

Обратное действие - прерывание поиска с возвратом - дает вызов предиката ***cut***, позволяющего выполнить отсечение всех альтернатив. Доказательство цели-отсечения всегда заканчивается успешно, и после этого не могут быть передоказаны цели, стоящие в утверждении до отсечения, включая головную цель.

Отсечение применяется для устранения бесконечных циклов, при программировании взаимоисключающих утверждений и при необходимости неудачного завершения доказательства цели.

Рассмотрим несколько вариантов правил определения абсолютного значения Y для некоторого числа X.

1 вариант.

```
abs(X,Y) :- X>=0, Y is X;
            Y is -X.
```

Для целевого утверждения

```
?- abs(5,X).
```

попытка сопоставления с первой альтернативой будет успешна: `X = 5`. Если продолжить поиск решений, то система выдаст неверный ответ `X = -5`, полученный при сопоставлении целевого утверждения со второй альтернативой.

2 вариант. Ограничим область применения второй альтернативы:

```
abs(X,Y) :- X>=0, Y is X;
            X<0, Y is -X.
```

Теперь система не будет выдавать лишних решений, но при положительных исходных числах по-прежнему будут проверяться обе альтернативы, хотя очевидно, что если $X \geq 0$, то вторая альтернатива заведомо не применима.

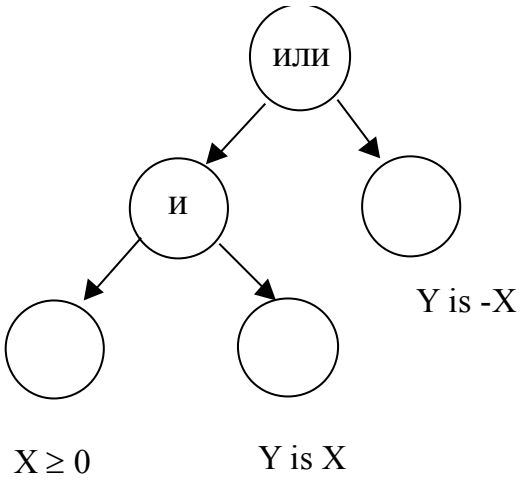
3 вариант. Для полного исключения возможности анализа заведомо лишней альтернативы, а следовательно, увеличения эффективности программы воспользуемся предикатом отсечения всех альтернатив данного определения.

```
abs(X,Y) :- X>=0, !, Y is X;
            Y is -X.
```

Предикат «!» запрещает проверку последующих альтернатив данного определения. После его удовлетворения невозможен выбор других альтернатив.

Выполнение программы на Прологе можно представить как построение ***И-ИЛИ дерева поиска решения*** и обход этого дерева. В И-ИЛИ дереве различают дав типа вершин И и ИЛИ. Ветви, сходящиеся к вершине И, считаются связанными конъюнктивно (логическое И), а ветви, сходящиеся к вершине ИЛИ – дизъюнктивно (логическое ИЛИ). Обход И-ИЛИ дерева осуществляется с крайней левой ветви (первого условия). Обнаружив определение с тем же именем, что и условие, Пролог достраивает дерево, используя те же правила, что и при интерпретации целевого утверждения: альтернативные варианты представляются ветвями, сходящимися к вершинам ИЛИ, а конъюнкция условий – ветвями, сходящимися к вершинам И. Поскольку факты условий не содержат, они интерпретируются в виде листьев дерева, тип их вершин не существенен.

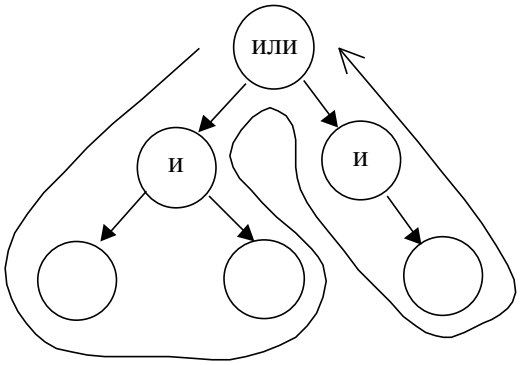
На рис.12 представлено И-ИЛИ дерево поиска решений для *1 варианта* определения.



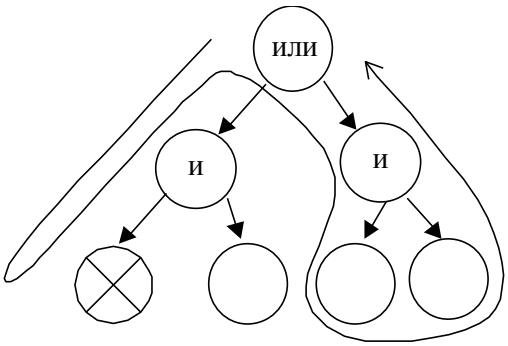

```
abs(X,Y):-  X>=0, Y is X;
            Y is -X.

?-abs(5,X).
```

рис. 12

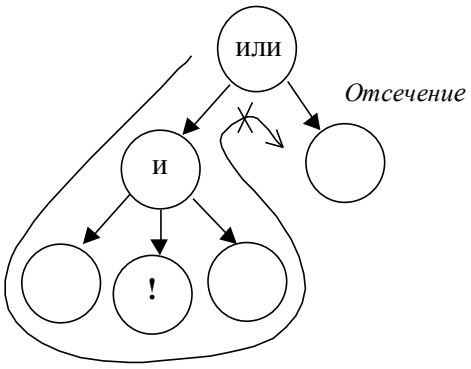
На рис.13 показана последовательность согласования целевых утверждений для различных вариантов программы. Сравнивая процессы выполнения программы, видно, что вторая альтернатива при $X \geq 0$ исключается из рассмотрения, при $X < 0$ процесс согласования цели не меняется.



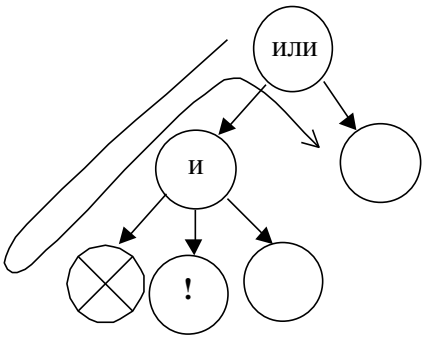
1 вариант. `?-abs(5,X).`



2 вариант. `?-abs(-5,X).`



3 вариант. `?-abs(5,X).`



3 вариант. `?-abs(-5,X).`

 - отвергнутая альтернатива

рис. 13

При использовании отсечения возможно возникновение побочных эффектов - исключение необходимых альтернатив и разрушение декларативного восприятия программы.

Встроенный предикат ***repeat*** всегда согласуется с базой данных и всегда может быть передоказан при возврате. Например, следующий запрос предлагает пользователю вводить значения до тех пор, пока не будет введено слово ‘stop’:

```
?-repeat,nl,write('Введите строку? '),read(X),X=stop.
```

Предикат `repeat` определен следующим образом:

```
repeat.
repeat:- repeat.
```

В Прологе имеется встроенный предикат отрицания – ***not***. Попытка доказательства предиката *not(X)* заканчивается успехом, если доказательство утверждения *X* заканчивается неудачей.

7.7 Динамическое изменение базы данных

При решении задач искусственного интеллекта достаточно часто возникает необходимость разработки программ, в которых заложена возможность изменения собственного текста, удаление и добавление отдельных фактов и правил. Встроенные предикаты позволяющие изменять текущее множество фраз программы:

assert(X) assertz(X)	добавление утверждения X (факта или правила) в конец базы данных, а также файл с текстом программы для некоторых версий Пролога
asserta(X)	добавление утверждения X в начало базы данных
retract(X)	удаление из базы данных (и из программы для некоторых версий) первого утверждения, которое унифицируется с X
retractall(X)	удаление всех утверждений, унифицируемых с X

Можно изменить содержание базы данных, сначала удалив факт, а потом вставив новую версию этого факта (или совершенно другой факт). Например, следующая программа изменяет факт, содержащий количество обращений, при каждом обращении счетчик увеличивается на единицу:

```
счетчик(0).
изменить_счет:- счетчик(X),
                X_нов is X + 1,
                retract(счетчик(X)),
                assert(счетчик(X_нов)).
```

7.8 Сферы применения Пролога

Существует ряд практических областей применения Пролога, вот некоторые из них:

- создание прототипов для любой прикладной программы. Первоначальная идея программы может быть быстро воплощена, а модель, на которой она основана, может быть реально проверена;
- построение динамических реляционных баз данных. Так как Пролог представляет реляционную БД как набор фактов, то он может быть использован как мощный язык запросов. Встроенный алгоритм поиска автоматически выбирает факты с верными значениями для известных параметров, присваивает значения неконкретизированным переменным, а алгоритм поиска с возвратом выдает все возможные решения по данному запросу;
- перевод с одного языка программирования на другой;
- понимание естественного языка;
- построение естественно-языкового интерфейса для существующего программного обеспечения, чтобы имеющиеся программные системы стали более доступными;
- создание экспертных систем;
- создание пакетов для задач символьной обработки - решения уравнений, дифференцирования и т.п.;
- доказательство теорем;
- создание программ искусственного интеллекта, которые используют дедуктивные возможности Пролога.

8. Формальные грамматики

Исходная информация, необходимая для решения задач управления может быть представлена в различных видах, в том числе и в символьной форме. Поэтому для систем искусственного интеллекта важное место занимают проблемы формальных грамматик.

Фраза на естественном языке представляет собой конечную последовательность элементов конечного множества слов. Не все сочетания слов возможны: одни имеют смысл, другие - не имеют. Очевидно, что это приводит к необходимости существования двух механизмов: механизма построения фраз, т.е. механизм порождения и механизма распознавания, предназначенного для определения корректности, т.е. понимания этих фраз.

Создать формализованную теорию естественного языка оказалось невозможным. Имеются лишь некоторые приближения, такие как грамматика Хомского. Для определения синтаксиса языков программирования наиболее употребительны ***КС-грамматики*** (*context-free; контекстно-свободные грамматики*).

КС-грамматики образуют разрешимые формальные системы. Это означает, что если язык L определен КС-грамматикой, то можно построить алгоритм распознавания фраз языка L.

В КС-грамматике определены: словарь T; некоторые последовательности слов (а это элементы замыкания T* словаря T) не являются “правильно составленными”. Язык L, который мы хотим описать, должен удовлетворять условию, что L является подмножеством T* как множество правильно построенных фраз.

Так осуществляется формализация логического языка. Известно, что в логике синтаксис языка определяется по инструкции, т.е. порождаются правильно составленные выражения языка. Сначала задается список исходных (базисных) выражений из языков высказываний и предикатов. Эти выражения соответствуют словам словаря различных синтаксических категорий (таких, как существительное, глагол, прилагательное) естественного языка и синтаксическим категориям формальных грамматик и языков.

В логике имеются правила построения, устанавливающие способы сочетания различных синтаксических категорий для образования более сложных выражений. Эти правила применяются рекурсивно. Правила аналогичны правилам естественного языка и правилам формальных грамматик. Пример, “подлежащие, сказуемое и определение - есть фраза”, отражает правила естественного языка. Фразы, записанные с помощью формальных грамматик, можно считать приближениями фраз естественного языка. Продукции формальных грамматик можно интерпретировать как логическую формализацию правил естественного языка, а фразы как подмножество логических формул.

Итак, формальные языки и грамматики можно рассматривать с одной стороны как результат, получаемый при формализации естественных языков и грамматик, а с другой стороны как частный случай логических языков и их синтаксисов.

Для определения языка, или естественного, или программирования, необходимо задать множество правил (продукции), образующих грамматику. Тогда, можно осуществить грамматический анализ фразы и описать структуру.

фраза $\rightarrow$ *глагол, группа_сущ;*

Для формирования фразы взять слово синтаксической категории - глагол и поставить за ним последовательность слов синтаксической категории - группа_сущ.

группа_сущ $\rightarrow$ *местоимение, прилагательное, существительное;*

группа_сущ $\rightarrow$ *местоимение, прилагательное, существительное, предлог, группа_сущ;*

глагол $\rightarrow$ *“читать”.*

В данном случае 1-3 - правила; 4 - пример факта.

КС-грамматика - гибкая по построению.

Формальное определение КС-грамматики: G = (V,T,P,S)

V - конечное множество переменных (синтаксических категорий);

T - конечное множество (непересекающееся с V) слов (терминальные символы);

P - конечное множество правил продукции типа A$\rightarrow$ L, где A $\in$ V, а L - это последовательность символов, принадлежащих V $\cup$ T;

S - особая переменная (фраза), которая называется исходным символом (начальной переменной).

Язык строится с помощью грамматики и интерпретации каждой из его продукции как правила переписывания или правила подстановки.

Полезно представлять порождаемые продукциями фразы в виде синтаксических деревьев.

Заметим, что если для формальных дизъюнктов не важен порядок расположения элементов, то для естественного языка - порядок важен. Формализация этого порядка производится с помощью списков и перехода к ОК-грамматике, т.е. к грамматике определенных клауз (Definiue Clause Grammar , DCG).

9. Элементы теории нейронных сетей

Нейронные сети как одно из направлений ИИ. Модель нейрона.Классификация в НС

Нейронные сети (НС) представляют собой одно из направлений развития ИИ. Моделирование и искусственная реализация нейронных сетей мозга человека и высших животных постоянно интересовало исследователей. Для объяснения мыслительных процессов выдвигались самые разнообразные идеи и теории - от бионических до синергетических. Например, для объяснения феномена памяти используются различные теории: ассоциативная теория (теория выдвинутая еще Аристотелем), условно - рефлекторная теория (Павлов), химическая теория и теория нейронных сетей.

Нервная система человека состоит из элементов, называемых нейронами. Поразительна сложность соединения нейронов. Около 10^{11} нейронов участвуют примерно в 10^{15} передающих связях. Уникальной способностью нейрона является свойства принимать, обрабатывать и передавать электрохимические сигналы по нервным путям, которые образуют коммуникационную систему мозга.

Считают, что биологический прототип нейрона состоит из тела самой клетки, дендритов и аксона (выходной отросток). Дендриты идут от тела нервной клетки к другим нейронам, где они принимают сигналы в точках соединения, именуемых синапсами. Принятые синапсом входные сигналы подводятся к телу нейрона, где сигналы суммируются. При этом одни входы стремятся возбудить нейрон, а другие тормозить. Если суммарное возбуждение в теле нейрона превосходит некий порог, то нейрон возбуждается, что означает передачу по аксону сигнала другим нейронам.

Об искусственных нейронных сетях, как объекте научного познания, стали говорить еще в 40-е годы. Первое систематическое исследование было предпринято Мак-Каллоком и Питтсом. Появляются на основе нейронных сетей системы для распознавания образов - персептроны. Модель нейрона Мак-Каллока-Питса стала базовой для построения персептрона Розенблатта.

Но искусственный нейрон только в первом приближении имитирует функционирование биологического нейрона.

На рис. 14 представлена модель искусственного нейрона. На вход искусственного нейрона поступает совокупность сигналов $\mathbf{x_1}$, $\mathbf{x_2}$, ... $\mathbf{x_n}$, в совокупности образующие вектор **X.** Это те сигналы, которые поступают в синапсы билогического нейрона. Каждый вход умножается на свой весовой коэффициент ($\mathbf{w_1}$, $\mathbf{w_2}$, ..., $\mathbf{w_n}$)

аналогичный синаптической силе. Все произведения суммируются, в результате можно оценить уровень активации нейрона. На суммирующем блоке (**E)** алгебраически складываются взвешенные входы, и в результате имеем

NET= XW,

∑ где **W** - вектор весовых коэффициентов; **X** - вектор входного сигнала;

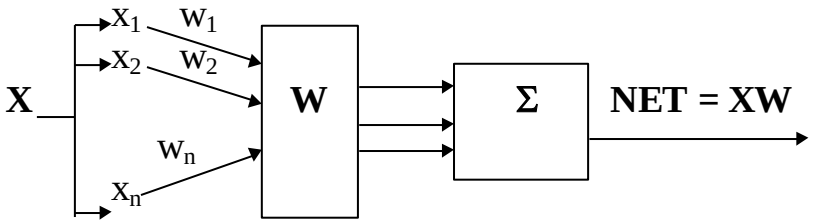


Рис. 14 Искусственный нейрон

Для расширения спектра моделируемых функций необходимо сигнал **NET** преобразовать Для этого вводятся активационные функции. В общем виде обозначим эту функцию - **F**. В результате функция **F** преобразует сигнал **NET** в выходной сигнал **OUT** (рис. 15).

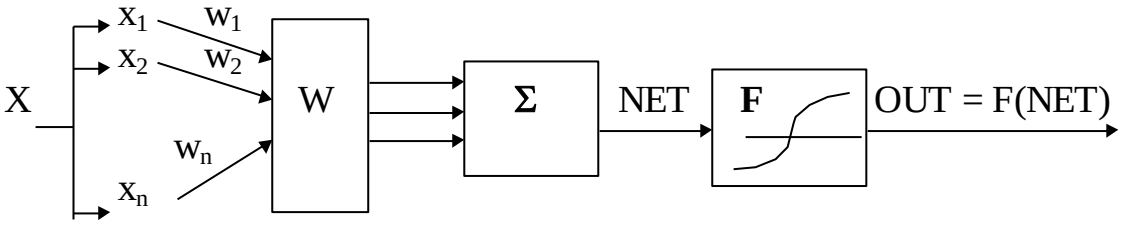


Рис. 15

Реализации функции **F** весьма разнообразны. Но наибольшее распространение получила S-образная функция. Также эта функция называется логистической функцией, сигмоидальной функцией, иногда - сжимающей функцией (так как она преобразовывает любой входной сигнал в конечный выходной). Эта нелинейная зависимость имеет вид:

OUT = 1/(1+exp(- NET))

Центральная область логистической функции имеет большой коэффициент усиления и позволяет решить проблему обработки слабых сигналов, а для больших возмущений на положительном и отрицательном концах имеется незначительное усиление.

Реально в нейронных сетях меняются коэффициенты усиления и структура. Сила нейронных вычислений заключается в соединениях нейронов в сетях.

Различают нейронные сети с обратными связями и без обратных связей. Сети без обратных связей называют также сетями прямого распространения; сети с обратными связями могут обладать свойствами кратковременной памяти.

Нейронные сети разделяются на:

-однослойные искусственные нейронные сети (рис. 16);

-многослойные искусственные нейронные сети.

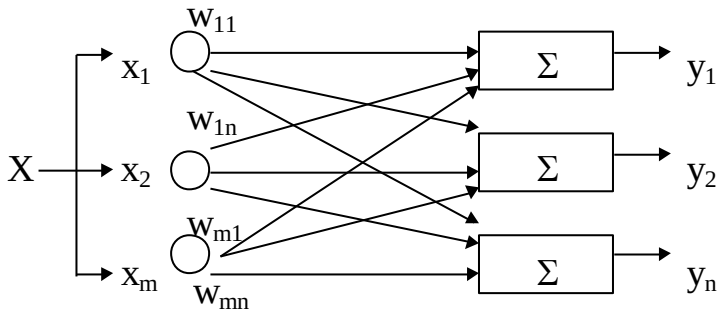


Рис. 16 Однослойные интеллектуальные нейронные сети.

Для однослойных НС:

$$\mathbf{N = XW},$$

где **N** - выходной вектор, а в матрице **W** число строк m -число входов; число столбцов n - число выходов или нейронов.

Многослойные, например, двухслойные интеллектуальные нейронные сети эквивалентны однослойным, только тогда единственный слой имеет весовую матрицу равную произведению двух весовых матриц двухслойной НС. Пример, пусть массив весовых коэффициентов второго слоя-**К.** Тогда результирующий вектор для двухслойной НС равен **N = (XW)K .** Так как умножение матриц ассоциативно, то это соответствует однослойной НС, но с матрицей **WK.** То есть **N = X(WK)** или **N= X W_1 , где W_1=WK.**

Многослойные сети могут привести к увеличению вычислительной мощности лишь тогда, когда активационные функции нелинейны.

Важное место в НС занимает обучение. Суть обучения в том, чтобы на основе некоторой статистической информации настроить сеть таким образом, чтобы предъявление букета входных сигналов, вызывало адекватное значение выходного сигнала. Получение желаемой функции осуществляется путем последовательного

предъявления входных векторов с одновременной подстройкой весов в соответствии с некоторым алгоритмом. В результате веса сети становятся такими, что возникает требуемое соответствие между предъявляемыми входными векторами и получаемыми выходными.

Различают алгоритмы обучения НС с учителем и без учителя. В обучении с учителем для каждого входного вектора существует целевой выходной вектор. Комбинация входного и выходного вектора называется обучающая пара. Предъявляется входной вектор, вычисляется выходной, последний сравнивается с целевым, на основании этого веса настраиваются таким образом, чтобы минимизировать ошибку.

Обучение без учителя в большей степени соответствует биологической системе и обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные сигналы при предъявлении достаточно близких входных векторов. Обучающее множество состоит только их входных векторов.

Применение нейронных сетей многообразно. Например, нейронные сети активно используются для решения задач классификации, задач диагностики. Кроме того нейронные сети являются теоретической и практической альтернативой алгоритмическому программированию и последовательной организации вычислительного процесса традиционных цифровых компьютеров с фон-неймановской архитектурой.

Литература

1. Амамия М., Танака Ю. Архитектура ЭВМ и искусственный интеллект. Пер. см японс.- М.: Мир, 1993.- 400с.
2. Братко И. Программирование на языке Пролог для искусственного интеллекта. Пер. с англ. - М.: Мир, 1990.-560с.
3. Григорьев Л.И., Арабаджи М.С., Гасымов И.Т. Экспертные системы и их применение (на примере нефтегазовой геологии). - М.: ИРЦ “Газпром”,1993. - 69с.
4. Григорьев Л.И., Ионкин С.П. Системы основанные на знаниях и их применение в нефтегазовой промышленности. ВНИИЭгазпром. Серия: в помощь экономическому образованию. Обз. инф. вып. 11-12. 1989.-56с.
5. Иванова Г.С., Тихонов Ю.В. Введение в МПролог. М: изд. МГТУ,- 1990. 152 с.
6. Ин Ц., Соломон Д. Использование Турбо-Пролога. Пер. с англ.- М.: Мир. 1993.- 608с.
7. Ковальски Р. Логика в решении проблем: Пер. с англ.- М.Мир. Наука. Гл.ред. физ.- мат. лит., 1990.- (Пробл. искусст. интеллекта.) - 280с.
8. ЛевинР., Дранг Д., Эделсон. Практическое Введение в технологию ИИ и ЭС с иллюстрациями на Бейсике. М. Финансы и статистика 1990.
9. Логический подход к искусственному интеллекту: от классической логики к логическому программированию. Пер. с франц. / Тейз А.,Грибомон П., Луи Ж. и др.- М.: Мир, 1990.- 432с.
10. Лорьер Ж.-Л. Системы искусственного интеллекта. Пер. с франц. - М.: Мир, 1991.- 588с.
11. Малпас Дж. Реляционный Пролог и его применение Пер. с англ. Наука. 1990.-464с.
12. Нильсон Н. Принципы искусственного интеллекта. Пер. с англ. - М.: Радио и связь. 1985.-376с.
13. Ойхман Е.Г., Попов Э.В. Реинжиниринг бизнеса: Реинжиниринг организаций и информационные технологии. М.: Финансы и статистика.1997.-336с.
14. Осуга Е. Обработка знаний. Пер. с японск. - М.: Мир, 1989.- 293с.
15. Попов Э.В., Фоминых И.Б., Кисель Е.Б., Шапот М.Д. Статические и динамические экспертные системы. Учебное пособие. М.: Финансы и статистика. 1996-320с.

16. Поспелов Д.А. Ситуационное управление: теория и практика.- М.: Наука.-Гл. ред. физ.-мат. лит., 1986- 288с.

17. Представление и использование знаний. Пер. с японск. /Под ред. Х.Уэно, М.Исидзука.- М.: Мир.1989.-220с.

18. Приобретение знаний. Под ред. С. Осуга, Ю. Саэки. Москва, Мир 1990.-304

19. Статьи и материалы, предоставленные фирмой Gensym (США).

20. Стерлинг Л.,ШапироЭ. Искусство программирования на языке Пролог. Пер.с англ. М.,Мир. 1990.- 236с.

21. Уоссермен Ф. Нейрокомпьютерная техника: теория и практика. Пер. с англ. - М.: Мир, 1992.-240с.

22. Уотермен Д. Руководство по экспертным системам. Пер. с англ. - М.: Мир,1989.-388с.