Головчинер М.Н.

ВВЕДЕНИЕ В СИСТЕМЫ ЗНАНИЙ

КУРС ЛЕКЦИЙ

Томск 2011

1. ПОНЯТИЕ СИСТЕМЫ ЗНАНИЙ

1.1.Понятие интеллектуального интерфейса

Архитектура **автоматизированного банка данных** может быть представлена следующей схемой (рис. 1.).

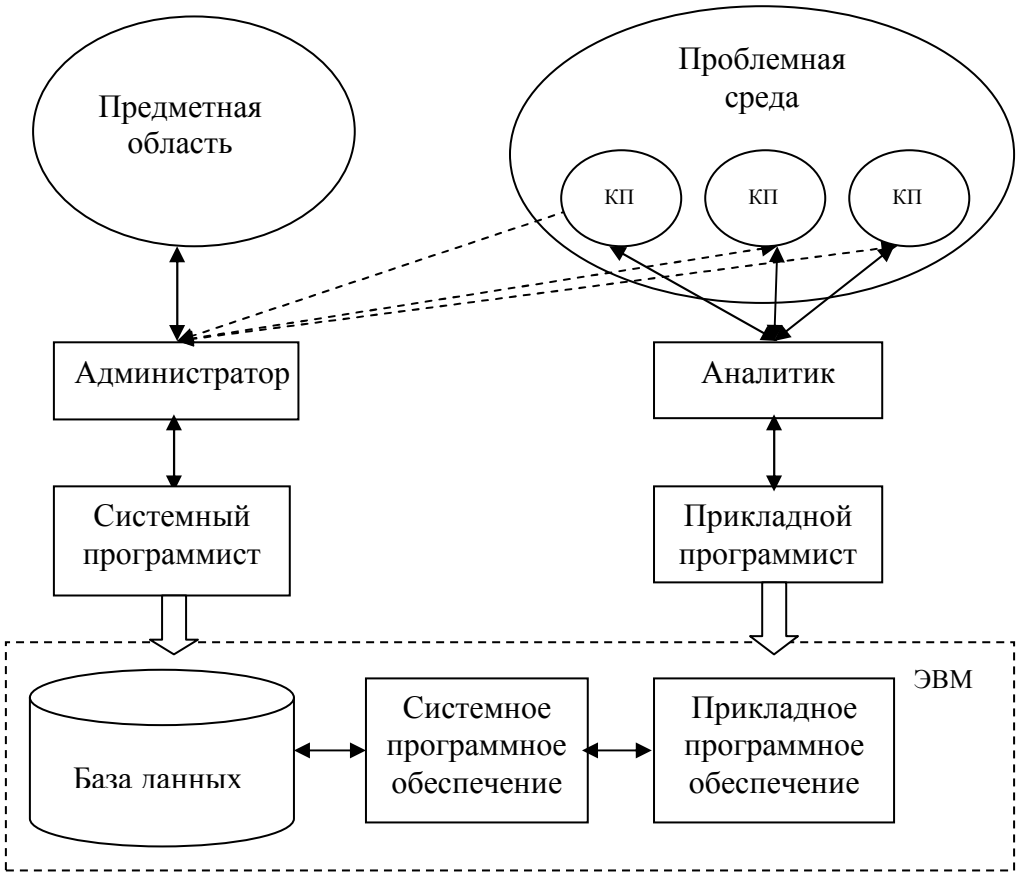


Рис.1. Архитектура автоматизированного банка данных

Технология решения задач в рамках такой информационной системы предполагает использование **посредников** (аналитиков и прикладных программистов) между конечными пользователями и ЭВМ, что приводит к повышению субъективного фактора при создании приложений. Действительно, качество работы программы полностью зависит от того, как аналитик понял задачу, сформулированную конечным пользователем, от адекватности и эффективности выбранных им моделей решения, от квалификации прикладного программиста в написании программного кода (реализации моделей).

Существенное снижение роли субъективной составляющей при решении задач в рамках интеллектуальных систем достигается путем исключения промежуточных звеньев между конечным пользователем и информационной системой. Ниже (рис.2.) представлена общая архитектура **автоматизированного банка знаний**.

Конечный пользователь общается с системой через терминал. Между пользователем и банком информации присутствует новый посредник - Интеллектуальный Интерфейс (ИИ).

ИИ представляет собой программно-алгоритмический комплекс, автоматизирующий основные функции аналитика и прикладного программиста.

К функциям аналитика здесь отнесем:

- общение с конечным пользователем при анализе постановки задачи;

- общение с конечным пользователем при интерпретации результатов полученного решения;
- перевод описания задачи на формализованный язык алгоритмов (выбор модели решения).

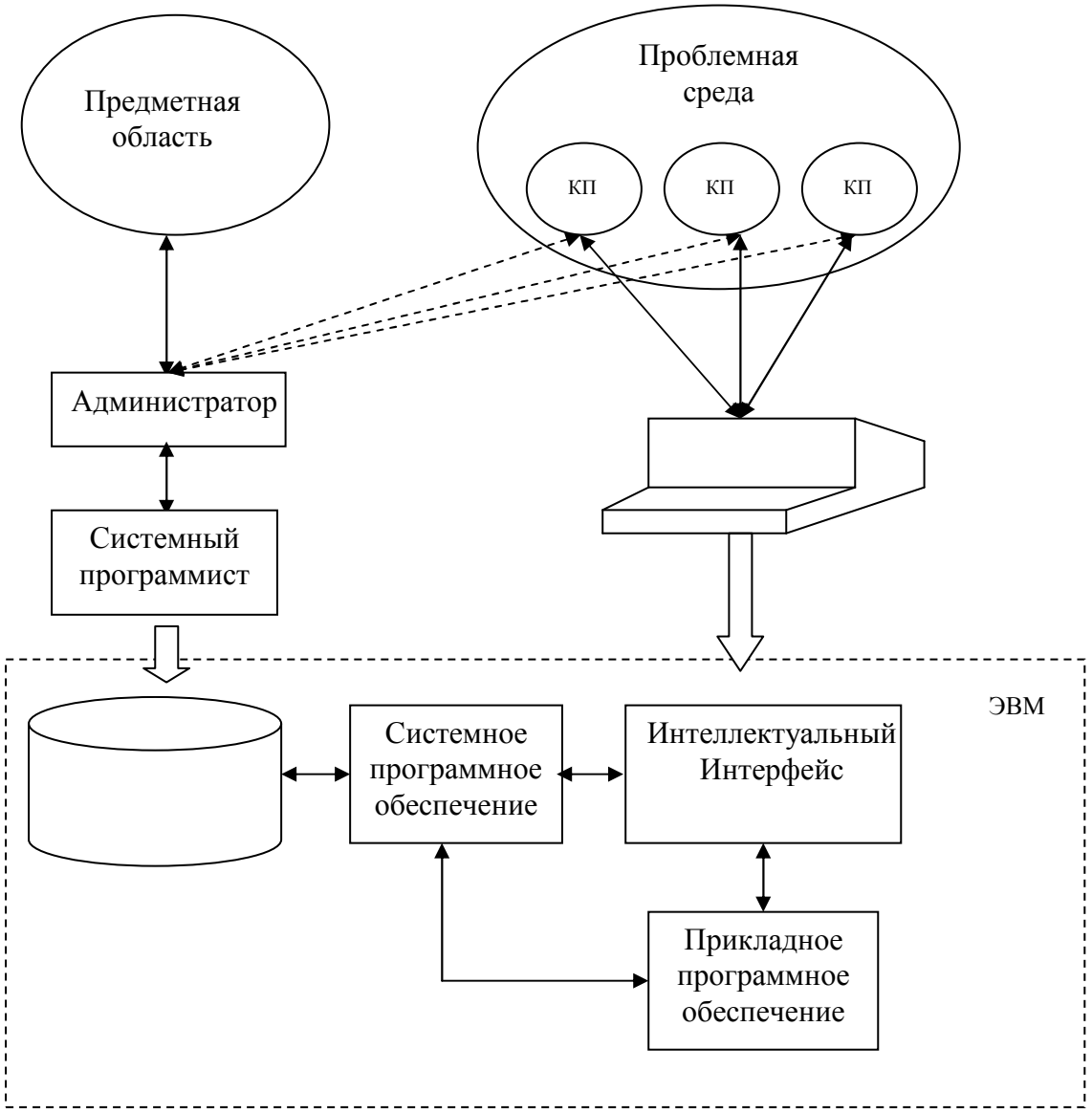


Рис.2. Общая структура архитектура автоматизированного банка знаний

Функции прикладного программиста ИИ выполняет на основе библиотеки программных модулей, из которых синтезируются прикладные программы, обеспечивающие решение поставленной информационной задачи конечным пользователем.

Чтобы ИИ мог выполнить возлагаемые на него функции по общению с пользователем, «пониманию» постановки задачи, синтезу программ, обеспечивающих решение задачи, представление полученных результатов в виде, удобном для восприятия, необходимо предоставить ИИ следующую информацию:

- знания о закономерностях, существующих в предметной области (ПО), и позволяющие выводить новые факты (имеющие место в данном состоянии ПО, но незафиксированные в явном виде в базе данных (БД);
- знания о проблемной среде, в которой работает конечный пользователь;
- знания о структуре и содержании БД;
- лингвистические знания, обеспечивающие «понимание» входного языка.

АБИ, имеющий в своем составе ИИ, будем называть **банком знаний**.
Ниже приведена общая структура Интеллектуального Интерфейса (рис.3.).

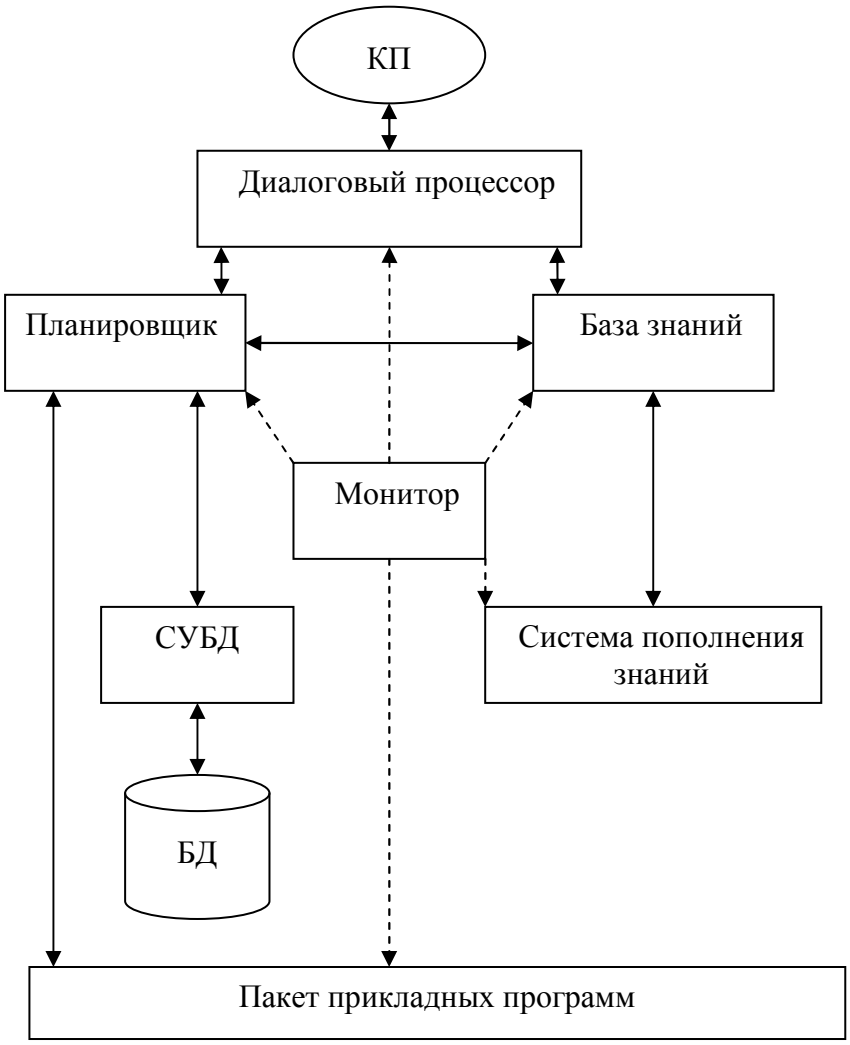


Рис.3. Общая структура Интеллектуального Интерфейса

Диалоговый (лексический) процессор - переводит сообщения конечного пользователя с естественного языка во внутреннее представление. Диалоговый процессор использует знания о входном языке, хранящиеся в базе знаний.
Планировщик (планирующая система) - преобразует поступающее на его вход формализованное описание исходной задачи в рабочую программу. Представляет собой специальный комплекс программ, во время работы постоянно контактирующих с базой знаний.
База знаний - хранит знания о предметной области и проблемной среде. Из базы знаний планировщик получает информацию:
о проблемной среде и способах решений задач;
о способе формирования рабочей программы;
о возможностях автоматического синтеза программ из набора базовых программных модулей, имеющихся в базе знаний.
Из **базы данных** через СУБД извлекается вся необходимая информация для выполнения синтезированной прикладной программы. Данные, получаемые из БД, выступают в этом случае как входные аргументы прикладной программы.
При отсутствии необходимой информации в базе данных и базе знаний знания качественного характера могут быть получены с помощью подсистемы пополнения знаний, которая реализует логические алгоритмы синтеза новых знаний. Кроме того, данная подсистема обеспечивает

автоматическое пополнение и модификацию знаний, исходя из результатов взаимодействия с конечными пользователями, то есть в соответствии с изменениями проблемной среды.

Таким образом, Искусственный интеллект - это одно из направлений информатики, целью которого является разработка аппаратно-программных средств, позволяющих пользователю-непрограммисту ставить и решать свои, традиционно считающиеся интеллектуальными задачи, общаясь с ЭВМ на ограниченном подмножестве естественного языка.

1.2.От данных к знаниям

Данные - это отдельные факты, характеризующие объекты, явления, процессы ПО, а также их свойства.
Знания - это закономерности ПО (принципы, связи, законы), полученные в результате практической деятельности и профессионального опыта, позволяющие специалистам ставить и решать задачи в этой области. Знания - это хорошо **структурированные данные**, или **данные о данных**, или **метаданные**.

Особенности знаний

1. **Внутренняя интерпретируемость**. Каждая информационная единица должна иметь **уникальное имя,** по которому система находит ее, а также отвечает на запросы, в которых это имя упомянуто.
   При переходе к знаниям в память ЭВМ вводится информация о некоторой **протоструктуре** информационной единицы (например, специальное машинное слово, в котором указано, в каких разрядах хранятся сведения о фамилиях, годах рождения и т.д.). Все современные СУБД обеспечивают реализацию внутренней интерпретируемости всех информационных единиц, представляющих содержимое БД, поддерживая ведение Словарей-справочников.
2. **Структурированность**. Информационные единицы должны обладать **гибкой структурой**. Для них должен выполняться «принцип матрешки», то есть, рекурсивная вложенность одних информационных единиц в другие. Другими словами, должна существовать возможность произвольного установления между отдельными информационными единицами иерархических отношений типа **«часть-целое», «род-вид», «элемент-класс».**
3. **Связность**. В информационной базе между информационными единицами должна быть предусмотрена возможность установления связей различного типа. Прежде всего, эти связи могут **характеризовать отношения** между информационными единицами. Семантика отношений может носить **декларативный** (**описательный**) или **процедурный** характер (см. далее классификацию знаний).

Примеры. Две и более информационных единицы могут быть связаны отношением:

- «**одновременно**» - временная связь (декларативное знание);
- «**причина-следствие**» - причинно-следственная (каузальная) связь (декларативное знание);
- «**быть рядом**» - пространственная связь (декларативное знание);
- «**аргумент-функция**» - функциональная связь (процедурное знание).

Очевидно, что между информационными единицами могут устанавливаться и иные связи, например, определяющие порядок выбора информационных единиц из памяти или указывающие на их несовместимость в одном описании.
Особенности 1÷3 позволяют ввести общую модель представления знаний - **семантическую сеть**.

4. **Семантическая метрика.** На множестве информационных единиц в некоторых случаях полезно задавать отношение, характеризующее ситуационную близость информационных единиц, то есть, **силу ассоциативной связи** (**отношение релевантности**) между информационными единицами. Отношение релевантности

позволяет находить знания, близкие к уже найденным. Такое отношение дает возможность выделять в информационной базе некоторые типовые ситуации.

5. **Активность**. С момента появления ЭВМ и разделения используемых в ней информационных единиц на данные и команды создалась ситуация, при которой **данные пассивны**, а **команды активны**. Все процессы, протекающие в ЭВМ, инициируются командами, а данные используются этими командами в случае необходимости. Для ИС эта ситуация неприемлема. Как и у человека, в ИС актуализация тех или иных действий способствуют знания, имеющиеся в системе. Таким образом, выполнение программ в ИС должно инициироваться текущим состоянием информационной базы. Появление в базе новых фактов или описаний событий, установление связей могут стать источником активизации системы.

Перечисление особенностей 1÷5 информационных единиц определяет ту грань, за которой данные превращаются в знания, а БД перерастает в БЗ.

Совокупность средств, обеспечивающих работу со знаниями, образуют систему управления базой знаний (СУБЗ).

1.3. Трансформация данных и знаний

При обработке на ЭВМ данные трансформируются, условно проходя следующие этапы:

1. Данные как результат измерений и наблюдений.
2. Данные на материальных носителях информации (таблицы, справочники).
3. Модели данных в виде диаграмм (ER-модель).
4. Данные в компьютере на языке описания данных.
5. Базы данных во внешней памяти ЭВМ.

Знания основаны на данных, полученных эмпирическим путем. Они представляют собой результат мыслительной деятельности человека, направленной на обобщение его опыта, полученного в результате практической деятельности.

При обработке на ЭВМ знания трансформируются аналогично данным:

1. Знания в памяти человека как результат мышления.
2. Материальные носители знаний (учебники, монографии).
3. **Поля знаний** - условное описание основных объектов ПО, их атрибутов и закономерностей, их связывающих.
4. Знания, описанные на языках представления знаний.
5. База знаний во внешней памяти ЭВМ.

1.4. Классификация знаний

1) Поверхностные и глубинные знания

Как правило, в предметной области можно выделить иерархии объектов трех видов: **структурная** (строение системы), **каузальная** (**причинно-следственая**, отражающая поведение системы) и **функциональная (**связь системы с внешним миром - метасистемой). Знания, связанные с этими уровнями, могут быть определены как **поверхностные** и **глубинные** согласно следующей схеме (рис. 4.).

Пример. Предметная область - компьютер.

1. **Структурная иерархия**. Рассмотрение компьютера как совокупности иерархически организованных физических компонент (до плат и далее микросхем). Очевиден иерархический путь поиска неисправности в компьютере. При этом метод локализации неисправности (поиск дефектной микросхемы) не требует наличия у проверяющего никаких знаний об устройстве компьютера или сведений из электроники. Не являются необходимыми и данные о функционировании блоков, требуются только **поверхностные знания** о структуре системы.
2. **Причинно-следственная иерархия, или модель поведения**. Функционирование компьютера описывается в терминах «причина-следствие». Например, на самом верхнем уровне иерархии нажатие клавиши на клавиатуре (или кнопки меню) должно привести к

выводу на печать. При отсутствии печати проверяется следующий уровень иерархии - работа принтера. Его действия тоже можно описать в терминах «причина-следствие». Ясно, что здесь требуются более глубокие знания, чем на структурном уровне.

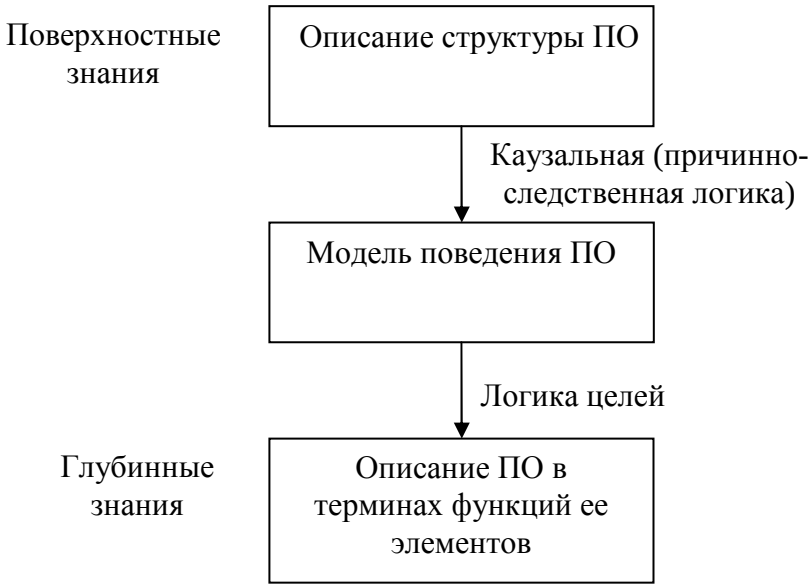


Рис. 4. Иерархии объектов ПО и уровни знаний

3. **Функциональная иерархия**. На самом высоком уровне абстракции компьютер рассматривается как иерархия модулей или подсистем, выполняющих определенные функции. Если не выполняется основная функция компьютера - преобразование поступивших входных данных в требуемые выходные в соответствии с заданным алгоритмом, на низлежащем уровне имеется целый спектр функций (ввод данных, обработка, вывод результатов), требующих анализа. Очевидно, что проведение этого анализа требует наличия **глубинных знаний**.

Таким образом, поверхностные знания включают представление о ПО только как об иерархии составляющих ее физических компонент. Эти знания представляют собой эвристики и некоторые закономерности, устанавливаемые опытным путем и используемые при отсутствии общих теорий и методов.

Глубинные знания отражают совокупность основных закономерностей, аксиом и фактов о конкретной ПО. Глубинные знания представляют наиболее общие принципы, в соответствии с которыми развиваются все процессы в ПО, и свойства этих процессов.

Замечания.

Каждый вид иерархии соответствует определенной точке зрения на ПО.

Можно рассматривать два типа иерархий: объективно существующие в ПО и отраженные в некотором ее представлении.

Не во всех ПО может быть установлена иерархия какого бы то ни было вида.

К любой ПО применимо деление знаний на **поверхностные** и **глубинные**. Это деление носит общий характер.

2) Знания как элементы семиотической системы

В общем виде знания представляются некоторой знаковой (семиотической) системой. В любой семиотической системе выделяют три аспекта:

- **синтаксический** - описывающий внутреннее устройство знаковой системы, то есть правила построения и преобразования знаковых выражений;
- **семантический** - определяющий отношения (связи) между знаками и свойства знаков, то есть задает смысл или значения конкретных знаков;
- **прагматический** - определяющий знак с точки зрения конкретной сферы его применения либо с точки зрения субъекта, использующего данную знаковую систему.

Соответственно, выделяют три типа знания:

- синтаксические - характеризуют синтаксическую структуру описываемого объекта или явления, не зависящую от смысла и содержания используемых при этом понятий;
- семантические - содержат информацию, непосредственно связанную со значениями и смыслом описываемых явлений и объектов;
- прагматические - описывают объекты и явления с точки зрения решаемой задачи.

3) Процедурные и декларативные знания

Информация, с которой имеет дело ЭВМ, разделяется на **процедурную**, которая представлена программами, выполняемыми в процессе решения задачи, и **декларативную**, описывающую данных, обрабатываемые программами.

Информационную базу в процессе выполнения программ образует содержимое оперативной памяти. Машинное слово является основной характеристикой информационной базы, так как каждое машинное слово хранится в одной стандартной ячейке памяти, снабженной индивидуальным именем - адресом (свойство интепретируемости знаний). Параллельно с развитием структуры ЭВМ происходило развитие информационных структур для представления данных. Появились способы описания данных в виде векторов и матриц, возникли списочные структуры. В настоящее время в языках программирования высокого уровня используются **абстрактные** типы данных, структура которых задается самим пользователем. Следующим шагом в организации работы с декларативной информацией стало появление Баз Данных.

Концепция знаний объединила в себе многие черты процедурной и декларативной информации.

1.5. Два подхода к организации вычислительного процесса

С двумя формами представления знаний в ЭВМ - процедурной и декларативной, их относительным весом в отображении проблемной среды, взаимосвязями и ролью в процессе обработки информации тесно связана постоянная эволюция программного обеспечения.

По отношению к роли процедур и данных при обработке информации на ЭВМ можно говорить о двух подходах:

- традиционном, ориентированном на процесс выполнения программы при решении задачи; то есть цель - выполнение программы;
- ориентированном на данные; то есть единственной целью процесса обработки является получение требуемых данных.

1) Традиционный подход

Процесс - это выполнение программы. В каждый момент времени процесс находится в определенном **состоянии**. Состояние процесса включает в себя всю информацию, необходимую для прерывания выполнения процесса с последующим его возобновлением. Состояние процесса содержит по меньшей мере следующую информацию:

- программу;
- индикацию (адрес) команды, которая должна выполняться следующей за текущей;
- значения всех программных переменных и данных;
- состояние всех используемых УВВ.

По мере протекания процесса его состояние меняется. В целом изменяющиеся параметры характеризуются **вектором состояния** процесса, по которому осуществляется контроль процесса обработки информации и управление им.

2) Технология ИИ

Контроль и управление процессом обработки информации в вычислительной системе осуществляется по состоянию всей совокупности данных, то есть наличию или отсутствию в этой совокупности тех или иных данных.

Таким образом, можно сформулировать требования к общей организации процесса обработки информации в системах ИИ:

Предоставление знаний. Решение любой задачи рассматривается как предоставление потребителю информации по его запросу нужного знания, удовлетворяющего некоторой спецификации на знания, содержащиеся в запросе.

Замечание. Под термином **потребитель знания** здесь понимается как конечный пользователь, выдавший запрос с терминала, так и любой процесс, обрабатывающий знания.

Информационная потребность. Необходимость для потребителя в том или ином знании. Определяется как отсутствие у потребителя информации, необходимой для решения подзадачи в составе общей задачи.

Примечание. В целом человеко-машинная система трактуется как структура, состоящая из компонентов, каждый из которых выполняет две функции:

- решение некоторой задачи по пришедшему извне запросу;
- формирование запроса к другим компонентам на недостающее для решения этой задачи знание.

Таким образом, задача представляется как множество подзадач, выполняемых различными компонентами системы, а весь процесс решения задачи распадается на множество процессов удовлетворений запросов различных компонентов. При этом конечный пользователь трактуется как некоторый конечный потребитель знаний, информационные потребности которого инициируют работу всех процессов.

Активность декларативных знаний. Ход решения задачи в любой момент времени оценивается по состоянию системы знаний системы, в которой процедурная часть остается постоянной, а декларативная постоянно меняется.

2. МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Модели представления знаний образуют две группы - **эвристические** и **логические** модели.

Эвристические подходы к представлению знаний носят в большей степени характер искусства, в их использовании превалирует интуиция, опыт и мастерство разработчика.

К числу эвристических моделей относятся **фреймовые модели** представления знаний и **семантические сети**.

В основе логических моделей лежит строгое понятие **формальной системы**, то есть изобразительные средства, лежащие в основе этих моделей, имеют теоретическое обоснование. Состав логических моделей представления знаний образуют **логические** и **продукционные** модели.

2.1. Фреймовые модели представления знаний

Теория фреймов излагается ее автором Минским (1974 г) в следующем виде:

«Когда человек сталкивается с новой ситуацией (или существенно меняет точку зрения на прежнюю задачу), он извлекает из памяти определенную структуру, называемую **фреймом**. Эту хранящуюся в памяти систему следует при необходимости привести в соответствие с реальностью путем изменения ее деталей».

Другими словами, фрейм - это структура данных, предназначенная для представления знаний о стереотипной ситуации, причем детали фрейма с изменением текущей ситуации могут меняться.

С каждым фреймом ассоциировано несколько видов информации; это, например, информация о том:

- как пользоваться данным фреймом,
- чего ожидать в следующий момент,
- что делать, если ожидания не подтвердятся,
- и т.д.

В целом фреймовая модель состоит из двух частей:

- набора фреймов, составляющих библиотеку внутри представляемых знаний,

- механизмов преобразования фреймов, их связывания и т.д.

2.1.1. Структура фрейма

В состав фрейма входят характеристики описываемой стереотипной ситуации (**слоты**) и их значения (**заполнители слотов**). Общая организация фрейма может быть представлена в следующем виде:

имя_фрейма: имя_слота$_1$, значение_слота$_1$

имя_слота$_2$, значение_слота$_2$

…….

имя_слота$_к$, значение_слота$_к$

Таким образом, слоты - это некоторые незаполненные подструктуры фрейма, заполнение которых приводит к тому, что данный фрейм ставится в соответствие некоторой ситуации, явлению или объекту.

Незаполненный фрейм (оболочка, образец, прототип фрейма), в котором отсутствуют заполнители слотов, называется **протофреймом**. Наличие такой оболочки позволяет осуществить процедуру внутренней интерпретации, благодаря которой данные в памяти системы не безлики, а имеют вполне определенный, известный системе смысл (обладают семантикой). Говорят, что протофрейм представляет **интенсиональное** описание.

Заполненный фрейм (экземпляр, пример прототипа) называется **экзофреймом**. Говорят, что экзофрейм соответствует **экстенсиональному** представлению протофрейма.

Более подробно структуру фрейма можно представить следующей схемой:

имя_фрейма

Имя_слота	Указатель наследования	Указатель атрибутов данны	Значения слота	Присоединенная процедура
Слот 1				
Слот 2				
…….				
Слот К				

Комментарии:

1. Каждый фрейм должен иметь **уникальное имя** в данной фреймовой системе.
2. Фрейм состоит из произвольного количества **слотов**. Некоторые из них обычно определяются самой системой для выполнения специфических функций, а остальные - самим пользователем.

Важным свойством фреймовых систем является то, что они могут представлять иерархические структуры, то есть реализовывать принцип **наследования**. Реализация механизма наследования основана на использовании системных слотов. Так, в число системных слотов входит слот, указывающий на фрейм-родитель и слот-указатель на дочерние фреймы.

3. **Указатель наследования.** Эти указатели касаются только фреймовых систем иерархического типа. Они показывают, какую информацию об атрибутах слотов во фрейме верхнего уровня наследуют слоты-потомки.

   Типичными указателями наследования являются:

   U (Unique) - уникальный. Фреймы-потомки должен иметь различные уникальные значения этого слота.

   S (Same) - такой же. Значение слота у всех потомков должно быть равным значению соответствующего слота фрейма-прародителя.

   R (Range) - интервал. Значение слота лежит в некоторых границах.

   O (Override) - игнорировать. Одновременное выполнение функций указателей U и S. При отсутствии значения слота у фрейма-потомка этим значением становится значение слота фрейма верхнего уровня (S), но допустимо и указание нового значения слота у фрейма-потомка (U).

4. **Указатель атрибутов (типов) данных**. К возможным типам данных относятся:
   a) Литеральные константы - INTEGER, REAL, BOOL, CHAR, …
   b) TEXT, LIST (список), TABLE, EXPRESSION, …
   c) LISP (присоединенная процедура),
   d) FRAME (фрейм)
   и др.

Таким образом, значением слота может быть объект произвольного типа и структуры (точнее, указатель на этот объект).

5. **Значение слота**.
   a) Тип значения должен совпадать с типом указателя атрибута данного.
   b) В качестве значений могут выступать выражения, содержащие обращения к функциям, имена таблиц, списков, других фреймов.
6. **Присоединенная процедура**. Выделяют два типа присоединенных процедур - **процедуры-слуги** и **процедуры-демоны**.
   Процедуры-слуги активизируются только при выполнении условий, определенных при создании фрейма.
   Процедуры-демоны активизируются при каждой попытке обращения к слоту. Среди разновидностей демонов можно отметить следующие:
   - «ЕСЛИ-НУЖНО» - активизируется, если в момент обращения к слоту его значение не было задано.
   - «ЕСЛИ-ДОБАВЛЕНО» - запускается при занесении в слот значения.
   - «ЕСЛИ-УДАЛЕНО» - запускается при стирании значения слота.

2.1.2. Использование фреймов

Пример использования системы фреймов. Пусть имеется иерархическая система протофреймов, описывающая понятие об отчете по выполнению работ по научной теме (рис. 5.).
Со слотами связаны следующие процедуры-демоны:
1.Слот **Автор**. Процедура «ЕСЛИ_ДОБАВЛЕНО»:
“Уведомить лицо из слота **Автор** , что ОТЧЕТ ПО ТЕМЕ из слота **Тема** ОБЪЕМОМ число страниц из слота **Объем** должен быть ПРЕДСТАВЛЕН к дате из слота **Дата представления”**.
2. Слот **Тема**. Процедура «ЕСЛИ ДОБАВЛЕНО»:
“Поместить в слот **Автор** ИМЯ РУКОВОДИТЕЛЯ ПРОЕКТА для ПРОЕКТ из слота **Тема”**
3. Слот **Дата представления**. Процедура «ЕСЛИ НУЖНО»:
“Поместить в слот **Дата представления** либо 31 марта, либо 30 июня, либо 30 сентября, либо 31 декабря (дата ближайшего конца квартала).”
Использование организованными таким образом знания может производиться по следующему сценарию.

1. Через терминал в систему поступает запрос:
“Необходим отчет о продвижении проекта по биологической классификации”.
2. В списке зкзотермов отчетов о продвижении система находит пустой узел с номером 12.
3. Интерфейсная программа анализирует этот запрос и вносит строку “Биологическая классификация в слот **Тема** узла 12.
4. Запускается процедура «ЕСЛИ ДОБАВЛЕ/НО» слота **Тема**. Эта процедура осуществляет поиск в базе данных системы ФИО руководителя проекта по биологической классификации; пусть это будет Иванов А.И. Процедура вписывает эти данные в слот **Автор** узла 12.
5. Запускается процедура «ЕСЛИ ДОБАВЛЕНО» слота **Автор**. Эта процедура начинает составлять сообщение для отправки его Иванову А.И.:
   “Иванову А.И. Пожалуйста, окончите отчет о продвижении работ по проекту Биологическая классификация к ……. Предполагаемый объем отчета равен ….. страницам.”
   В процессе формирования сообщения процедура обнаруживает пустым слот **Дата представления**.

Замечание. Формируемое сообщение представляет собой шаблон, в который вставляются значения слотов соответствующего экзотерма. Места вставок отмечены строками …..

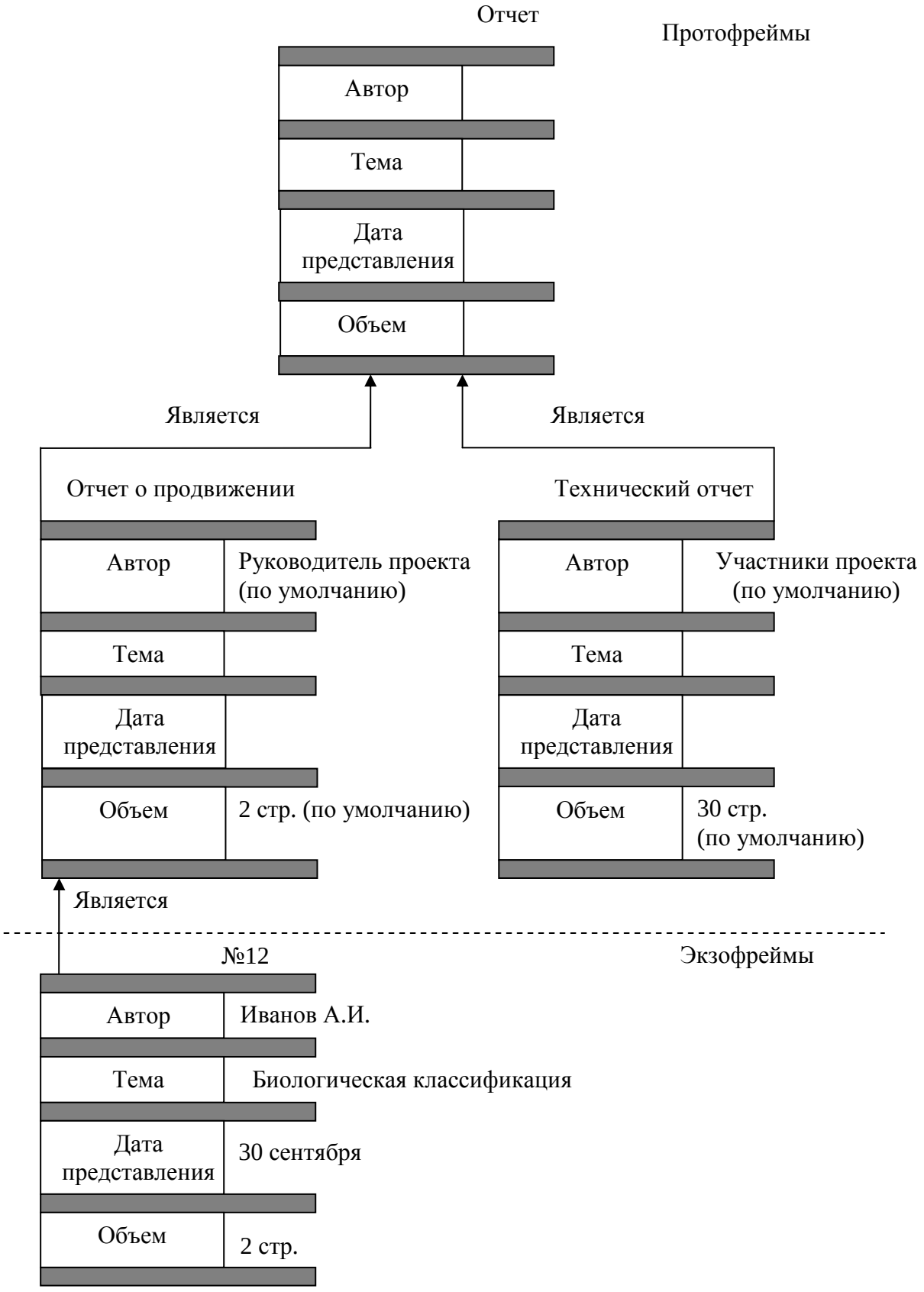
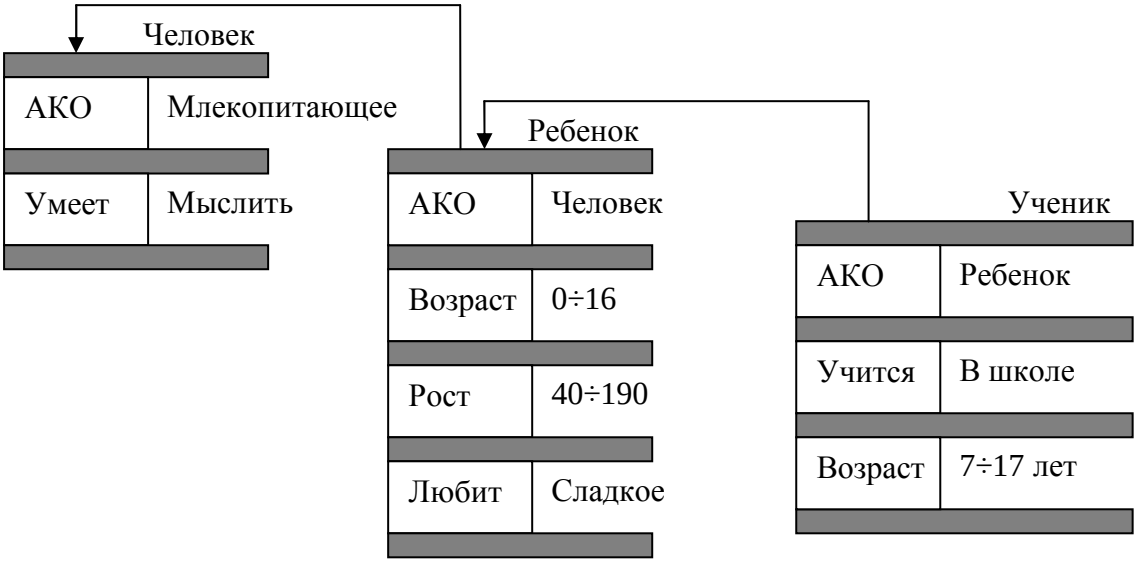


Рис. 5. Пример иерархической системы фреймов

2.1.3. Организация логического вывода на фреймах

Заметим, что наличие иерархических связей позволяет в рамках фреймовой системы организовать, используя свойство транзитивности, логический вывод.
Пример. Пусть имеется система фреймов.



Слот **АКО (**A Kind Of**)** - это системный слот, значением которого является ссылка на родительский фрейм, из которого неявно наследуются некоторые свойства фрейма-потомка.
Вопрос: “Любит ли ученик сладкое?”
Ответ: “Да”. Вывод сделан на основании наследования слота **Любит** фрейма **Ребенок**.
Вопрос: “Может ли ученик мыслить?”
Ответ: “Да”. Вывод сделан на основании наследования слота **Умеет** фрейма **Человек**.

2.1.4. Достоинства и недостатки фреймовых моделей

Достоинства

1. Представление знаний, основанное на фреймах, дает возможность хранить родовую иерархию понятий в Базе знаний в явной форме.
2. Принцип наследования позволяет экономно расходовать память, проводить анализ ситуации при отсутствии ряда деталей.
3. Фреймовая модель является достаточно универсальной., поскольку позволяет отобразить все многообразие знаний о реальном мире через:
   - фреймы-структуры, использующиеся для обозначения объектов и понятий (залог, вексель);
   - фреймы-роли (клиент, менеджер);
   - фреймы-сценарии (банкротство, собрание);
   - фреймы-ситуации (авария, рабочий режим устройства);
   - и др.
4. С помощью присоединенных процедур фреймовая система позволяет реализовать любой механизм управления выводом.

Недостатки

1. Относительно высокая сложность фреймовых систем, что проявляется в снижении скорости работы механизма вывода и в увеличении трудоемкости внесения изменений в родовую иерархию.

2. Во фреймовых системах затруднена обработка исключений. Наиболее ярко достоинства фреймовых систем представления знаний проявляется в том случае, если родовидовые связи изменяются нечасто и предметная область насчитывает немного исключений.

3. Разрозненные части информации, объединенные во фрейм, не могут быть выстроены в последовательность высказываний, иначе говоря, языки описания знаний во фреймовской модели не являются языками, родственными естественным, а ближе к изобразительным средствам.
4. Отсутствует специальный механизм управления выводом, поэтому он реализуется с помощью присоединенных процедур.

2.2. Сетевая модель представления знаний

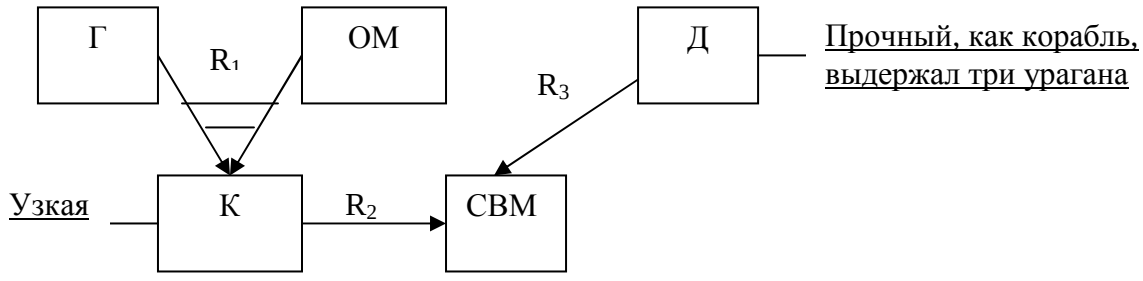
2.2.1. Общая структура семантической сети

В самом общем виде семантическая сеть есть множество вершин, каждая из которых соответствует определенному понятию, факту, явлению или процессу; а между вершинами заданы различные отношения, представляемые дугами.
Вершины помечены **именами** и **описателями**, содержащими нужную для понимания семантики вершины информацию.
Дуги также снабжены **именами** и **описателями**, задающими семантику отношений.
Формально сетевую модель S (семантическую сеть) можно задать в виде

$$\mathbf{S = <I, G_1, G_2, \ldots, G_N, R>},$$ где

I - множество информационных единиц, представленных вершинами сети;
$G_1, G_2, \ldots, G_N$ - заданный набор типов отношений между информационными единицами;
R - отображение, задающее между информационными единицами, входящими в I, связи из заданного набора типов связей.

Пример 1 С помощью семантической сети моделируется следующий отрывок литературного произведения (рис.6):
“Дом был построен на самом высоком месте узкой косы между гаванью и открытым морем. Построен он был прочно, как корабль, и выдержал три урагана. Его защищали от солнца высокие кокосовые пальмы, пригнутые пассатами, а с океанской стороны крутой спуск вел прямо от двери к белому песчаному пляжу, который омывался Гольфстримом.”
Введем систему понятий (информационных единиц):
Д - дом; **СВМ** - самое высокое место; **К**- коса; **Г** - гавань; **ОМ** - открытое море; **КП** - кокосовые пальмы; **С** - солнце; **КС** - крутой спуск; **ДВ** - дверь; **П** - пляж; **Гф** - Гольфстрим.
Построим фрагмент (А) семантической сети, соответствующий первым двум фразам текста.

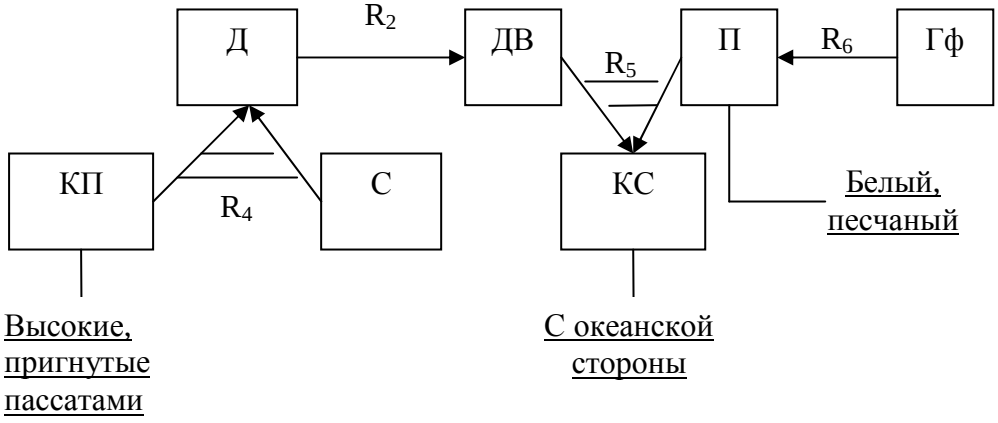


Где:
R_1 - отношение “быть между”;
R_2 - отношение “принадлежать”;
R_3 – отношение “находиться на”;
__ - подчеркнутый текст соответствует описателю свойств вершины.

Фрагмент (А)

Остальной части текста соответствует фрагмент (В), где:

R_4 - отношение “защищать от”;
R_5 - отношение “соединять”;
R_6 - отношение “омывать”.



Фрагмент (В)

Рис.6. Семантическая сеть, моделирующая литературный текст

Полное описание текста в виде семантической сети получится, если объединить фрагменты (А) и (В), соединив вершины, соответствующие понятию “Дом”.
Очевидно, что можно элементы сети, в частности, описатели вершин, представить и по-другому; например, для понятий “ураган” и “пассат” ввести отдельные вершины и включить их в общую структуру сети.
Пример 2. Семантическая сеть и грамматический разбор фраз естественного языка (рис. 7.).
Семантические сети успешно используются в научных работах по естественному языку для представления сложных грамматических предложений:
“Петр сказал Ирине, что он отдал Наталье подарок.”
Этот пример иллюстрирует тот факт, что узлы сети могут соответствовать не только понятиям-сущностям, но и понятиям-процессам (“Говорить”, “Давать”).

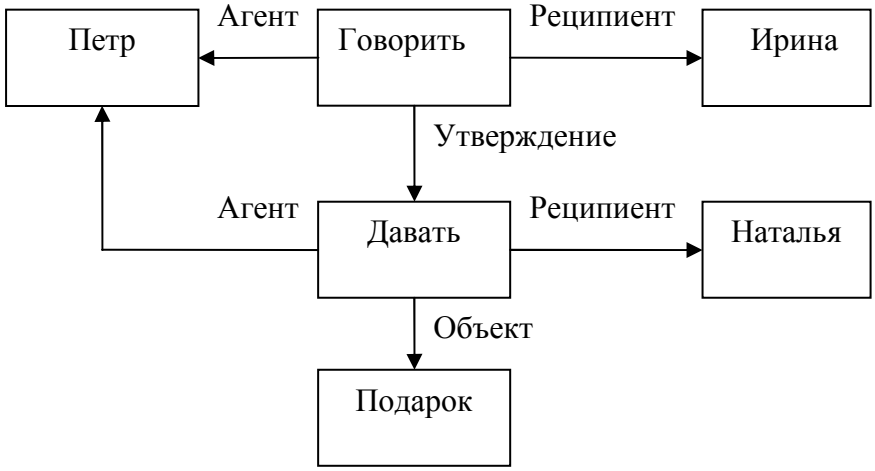


Рис.7 Семантическая сеть, моделирующая структуру фразы

2.2.2. Типы объектов и отношений в семантических сетях

Любая более или менее сложная модель любой предметной области отображает какие-либо **обобщенные**, **конкретные** или **агрегатные** объекты.

Обобщенный объект - некоторое известное и широко используемое в ПО понятие, представляющее определенным образом некоторый **тип** объектов.

Конкретный (индивидуальный) объект - выделенная одиночная сущность (**экземпляр** объектов некоторого типа).

Агрегатный (составной)объект - составлен тем или иным способом из других объектов, являющихся его частями.

Очевидно, что определение того или иного типа объекта условно и зависит от взгляда на ПО, другими словами, от проблемной среды.

Классификация отношений в семантических сетях

Классификацию отношений можно проводить по разным основаниям. Примеры таких классификаций приведены ниже.

По **количеству типов отношений** выделяют:

- однородные сети (с единственным типом отношений),
- неоднородные сети (с различными типами отношений).

Выделяют следующие **типы** отношений:

- бинарные (отношения связывают два объекта);
- N-арные (отношение связывает более двух обектов).

Наиболее часто в семантических сетях встречаются следующие **виды** отношений:

- иерархические («РОД-ВИД», «ЭЛЕМЕНТ-МНОЖЕСТВО», «ЧАСТЬ-ЦЕЛОЕ» и т.п.);
- функциональные («АРГУМЕНТ-ФУНКЦИЯ», а также связи, определяемые глаголами «влияет», «производит» и другими);
- количественные («больше», «меньше», и т.д.);
- пространственные («далеко от», «близко от», «над», «под» и т.д.);
- временные («раньше», «позже», «в течение»);
- атрибутивные («иметь свойство», «иметь значение»);
- каузальные (причинно-следственные;
- логические («И», «ИЛИ», «НЕ»);
- лингвистические;
- и др.

Проблема поиска решения в базе знаний типа семантической сети сводится к задаче поиска фрагмента сети, соответствующего некоторой подсети, отражающей поступивший в систему запрос.

В зависимости от смысловой нагрузки отношений семантические сети можно классифицировать по разным типам:

Классифицирующие сети. Используют отношения структуризации. Такие сети позволяют вводить в Базу Знаний иерархические отношения между информационными единицами.

Функциональные сети. Характеризуются наличием функциональных отношений (вычислительные модели).

Сценарии. Используют каузальные связи.

2.2.3. Виды семантических сетей

Иерархические сети. Являются важнейшим видом семантических сетей. Отметим два важных вида иерархических связей между информационными единицами.

1. Между двумя обобщенными объектами может существовать **родовая связь**. Наличие ее между обобщенными объектами А и В означает, что любой объект, отображаемый понятием В, отображается понятием А и существует такой объект, который отображается А, но не

отображается B (A более общее, чем B). Так, например, понятие «млекопитающее» является родовым для понятия «человек».

Видовая связь является обратной к родовой. Если объект A является **родом** объекта B, то объект B является **видом** объекта A.

Ясно, что родовое понятие не охватывает всех свойств видового; видовое понятие в общем случае богаче содержанием. С другой стороны, все свойства родового понятия присущи и видовому (наследование свойств).

2. Между обобщенным и конкретным объектами может существовать связь «является представителем» (или, что аналогично, «быть элементом класса»).

Свойства, присущие обобщенному объекту, характеризуют и любой конкретный объект-представитель. Таким образом, множество свойств конкретного объекта содержит в себе подмножество свойств, которыми он наделяется как представитель тех или иных обобщенных объектов.

Замечание. Один и тот же конкретный объект может рассматриваться как представитель нескольких обобщенных объектов в одной и той же предметной среде. Так, например, конкретный объект «Третьяковская галерея» является представителем двух обобщенных объектов - «картинная галерея» и «здание в Москве». Причем объект «картинная галерея» находится в родовидовой связи с обобщенным объектом «музей», а объект «здание в Москве» нет, так как музей может размещаться в нескольких зданиях и просто на открытом воздухе, и не всякое здание в Москве является музеем.

Этот пример показывает, что к выводам, основанным на комплексном использовании родовидовых связей и связей подчиненности, следует подходить осторожно.

Организация поиска в иерархической сети

Назовем **простой** семантическую сеть, у которой для вершин не определена внутренняя структура.

Одним из основных отличий иерархических сетей от простых семантических состоит в том, что иерархические сети можно разделить на подсети (пространства) и установить отношения не только между вершинами, но и между пространствами. Очевидно, что все вершины и дуги являются элементами, по крайней мере, одного пространства.

Замечание. Понятие пространства аналогично понятию скобок в математической нотации. Различные пространства, существующие в сети, могут быть упорядочены в виде дерева пространств, вершинам которого соответствуют пространства, а дугам - отношения «видимости».

Из пространства P_6 (пространство-потомок) видимы все вершины и дуги, лежащие в пространствах-предках P_4, P_2, и P_0, а остальные пространства «невидимы».

Отношение «видимости» позволяет сгруппировать пространства в упорядоченные множества - **перспективы**. Перспектива обычно используется для ограничения сетевых сущностей, «видимых» некоторой процедурой, работающей с сетью. Очевидно, что свойство «видимости» позволяет повысить эффективность операции поиска в сети (рис. 8.).

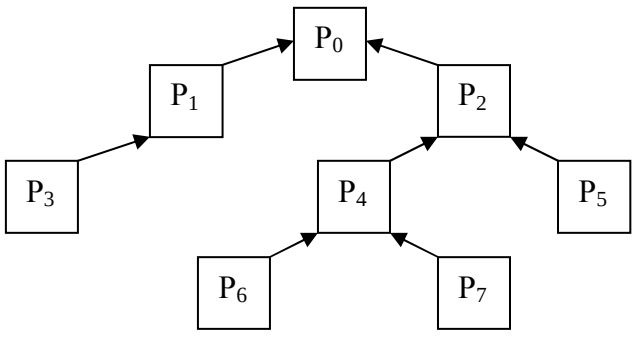


Рис.8. Пример множества **перспектив**

Организация вывода на сети. Рассмотрим следующий пример семантической сети.

Вопрос "Находится ли под наблюдением органов контроля Иванов А.И.?"
Очевидно, что ответ зависит от семантики связи «Держать пол наблюдением» (рис.9). Если это связь иерархическая, то ответ положителен однозначно на основании свойства транзитивности иерархических связей. В противном случае все зависит от действия процедуры, реализующей эту связь.

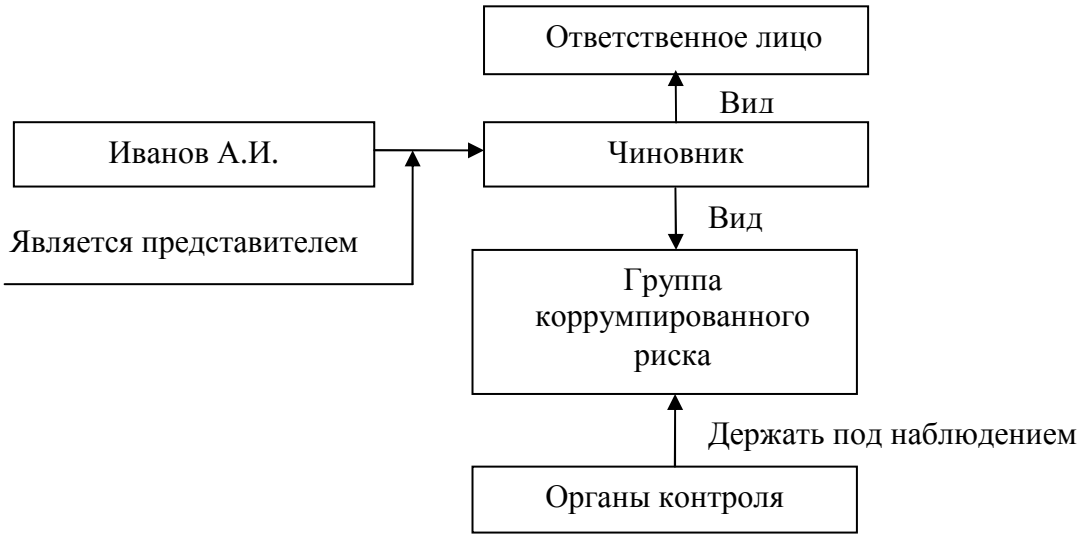


Рис.9. Пример организации вывода на сетях

Семантические сети и фреймы. Заметим, что в чисто сетевых моделях понятия **атрибута** объекта (вершины) как такового не существует, определение атрибутов объекта осуществляется через соответствующие отношения. Это:

- усложняет структуру сети;
- существенно затрудняет реализацию аппарата наследования.

Поэтому на практике вершины семантической сети представляются фреймами, а дуги - значениями соответствующих слотов.

Процедурные семантические сети. В целях введения единой семантики в семантические сети могут быть использованы **процедурные** семантические сети. В этом случае сеть строится на основе **класса** (понятия), а вершины, дуги (отношения) и процедуры представляются как объекты. Процедурами определяются все основные действия над вершинами и дугами (связями). В таких семантических сетях кроме декларативных возможно представление и процедурных знаний.

2.2.4. Достоинства и недостатки сетевой модели представления знаний

Достоинства:

1. Большие выразительные возможности, естественность и наглядность системы знаний, представленной графически.
2. Близость структуры сети, представляющей систему знаний, семантической структуре фраз естественного языка.
3. Данная модель более других соответствует современным представлениям об организации долговременной памяти человека.

Недостатки:

1. Громоздкость и неэффективность представления знаний только аппаратом семантической сети.
2. Сложность организации процедуры поиска нужного знания (как фрагмента сети).

2.3. Логические модели представления знаний

2.3.1. Описание предметной области. Понятие сигнатуры

Основная идея логического подхода состоит в том, чтобы рассматривать всю систему знаний, необходимую для решения прикладных задач и организацию взаимодействия ЭВМ с пользователем. Как совокупность фактов (утверждений). Факты представляются как формулы в некоторой логической системе. Система знаний отображается совокупностью таких формул; будучи представленная в ЭВМ, она образует Базу Знаний. Формулы неделимы и при модификации БЗ могут лишь добавляться и удаляться.

Описания ПО, выполненные в рамках логических языков, называются логическими (формальными) моделями представления знаний.

Для представления математических знаний в математической логике давно пользуются логическими формализмами - главным образом, **исчислением предикатов**, которое имеет ясную формальную семантику и операционную поддержку в том смысле, что для него разработаны механизмы вывода. Поэтому исчисления предикатов было первым логическим языком, который был применен для формального описания предметных областей, связанных с решением прикладных задач.

Инструментарий логических моделей представления данных рассмотрим на примере логического языка, называемого **многосортным исчислением предикатов первого порядка** (или **многосортной логикой первого порядка**).

В логических моделях, основанных на исчислении предикатов, типам объектов ПО отвечают, так называемые, **сорта**. Точнее, в логических моделях участвуют **имена** сортов, которые интерпретируются как типы сущностей.

Пример. Описание фрагмента ПО, связанного с учебным процессом.

Введем следующие имена сортов:

Студент, **Ном.зач.**, **Группа, ФИО, Дисциплина**, **Семестр**, **Оценка_Экзамен**, **Оценка_Зачет**, **Дата.**

Определим набор **функций** и **предикатов**:

(1) **ст.ном.зач.**: **Студент → Ном.зач.**
(2) **ст.гр.: Студент → Группа**
(3) **ст.фио: Студент → ФИО**
(4) **изучает: Студент → Дисциплина**
(5) **экз.оценка: Дисциплина → Оценка_Экзамен**
(6) **зач.оценка: Дисциплина → Оценка_Зачет**
(7) **проводится: Дисциплина → Семестр**
(8) **зачет: Дисциплина → T**
(9) **экзамен: Дисциплина → T**
(10) **≤ : Дата x Дата → T**
(11) **+ : Оценка_Экзамен x Оценка_Экзамен → Оценка_Экзамен**
(12) **код_студ : → Студент**
(13) **ном_зач : → Ном.зач**
(14) **фио : → ФИО**
(15) **ном_гр : → Группа**
(16) **наим_дисц : → Дисциплина**
(17) **оц_зач : → Оценка_Зачет**
(18) **оц_экз : → Оценка_Экзамен**
(19) **дата_нач : → Семестр**
(20) **дата_оконч : → Семестр**
(21) **ном_семестра : → Семестр**

Выражения (1) - (19) составляют **сигнатуру** и имеют следующий смысл.

1. (1) – (7) задают **структурные функции**, определяющие связи между сортами. Например, значением функции **ст.гр.,** примененной к объекту **e** сорта **Студент**, то есть

ст.гр.(e) , является объект сорта **Группа**. Соответствующая структурная схема для заданной ПО приведена на рис.10.

(8) – (10) задают логические связи, представленные **предикатами** – функциями, принимающими значения из множества T={0,1} («истина», «ложь»). Выражения (8) и (9) являются **одноместными** предикатами, определяющими отчетность по соответствующей дисциплине (сдача зачета или экзамена). Например, **зачет(e)=1** , где **e** – объект сорта **Дисциплина**, тогда и только тогда, когда **e** дисциплина, для которой предусмотрена сдача зачета.

Выражение (10) задает **двуместный** предикат (двуместное отношение «**≤»)**, заданный на декартовом произведении объектов сорта **Дата**. Этот предикат позволяет определить, например, корректность задания начала и окончания определенного временного интервала.

(11) задает **функциональную связь** - двуместную **функцию** «+», аргументами которой служат объекты сорта **Оценка_Экзамен** (очевидная интерпретация этой функции - сложение натуральных чисел). Очевидно, что данная функция необходима для подсчета, например, среднего результата сдачи сессии.

(12) – (21) задают связь типа «**иметь значение**», представленную нуль-местными функциями (множествоми констант), определяющими области значений, принимаемых свойствами объектов соответствующих сортов. Здесь:

код_студ – код студента (S),
ном_зач – номера зачетных книжек (S),
фио – ФИО студентов (S),
ном_гр – номера студенческих групп (S),
наим_дисц – наименования дисциплин (S),
оц_зач – результат сдачи зачета («зачтено», «не зачтено»),
оц_экз – оценка сдачи экзамена {3, 4, 5},
дата_нач – дата начала семестра (D),
дата_оконч – дата окончания семестра (D),
ном_семестра – номер семестра {1, 2, 3, 4, 5, 6, 7, 8,}.

Где:

S (строка символов) и D (дата) – типы соответствующих констант,
{…} – фиксированное множество значений.

Замечание. На возможные значения сорта могут быть наложены ограничения, например, не могут быть произвольными строки символов, задающих ФИО студента или номер группы, в которой он учится.

Замечание. Связь «**иметь значение**» также является структурной. На схеме рис.9 она опущена.

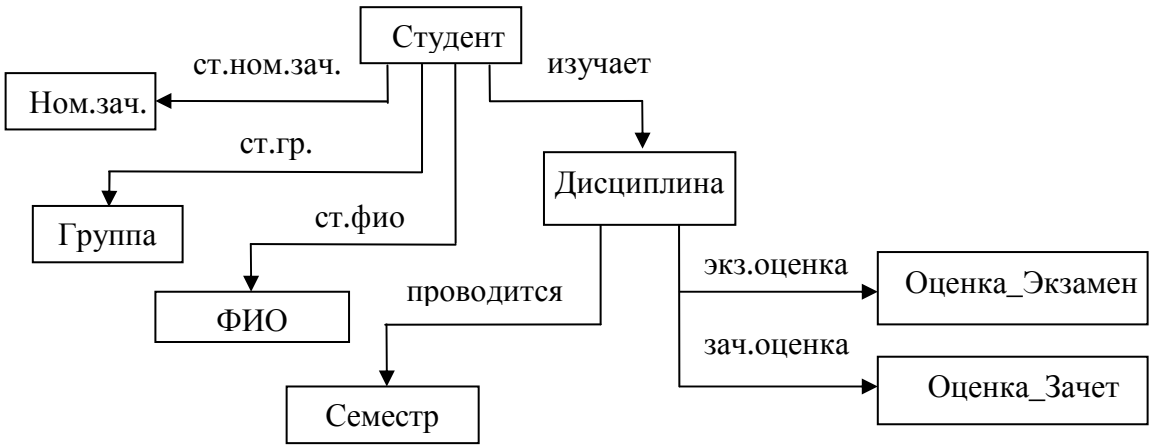


Рис. 10. Структурная схема фрагмента ПО «Учебный процесс»

Итак:

2. Сигнатура задает **структурные связи** в ПО.
3. Сигнатура задает при помощи предикатов **логические связи**.
4. Сигнатура задает **функциональные связи**.

В общем случае **сигнатурой** называется множество ∑ выражений вида
(1) f: $\mathbf{A_1 x A_2 x \dots x A_n \rightarrow B}$, где
Ai, B - сорта,
f - функция.
(2) **p**: $\mathbf{A_1 x A_2 x \dots x A_n \rightarrow T,}$ где
Ai - сорта,
T - множество, состоящее из двух логических констант {**true** (**истина)**, **false (ложь)**}, кодируемых, соответственно, **1** и **0**.
Предикат можно рассматривать также как многоместное отношение между объектами разных сортов: говорят, что объекты $\mathbf{e_1, e2, \dots e_n}$ находятся в отношении **p**, что записывается как $\mathbf{p(e_1, e_2, \dots e_n)}$ тогда и только тогда, когда $\mathbf{p(e_1, e_2, \dots e_n)=1}$ **(истина).**
Замечание. Для двуместных предикатов чаще используется **инфиксная** запись: вместо $\mathbf{p(e_1, e_2)}$ пишут $\mathbf{(e_1\ p\ e_2)}$, то есть, не $\mathbf{\leq (\ e_1, e_2),}$ а $\mathbf{(e_1 \leq e_2).}$

2.3.2. Построение формул

Пусть задана произвольная сигнатура ∑. Произвольная правильно построенная формула (ППФ) в этой сигнатуре формируется по следующим правилам.

A) Построение **термов**. Термы строятся по следующим правилам.

П1. Переменная, принимающая значения из сорта А, есть терм сорта А.

П2. Если сигнатура ∑ содержит выражение

$$\mathbf{f: A_1 x A_2 x \dots x A_n \rightarrow B}$$

и $\tau_1, \tau_2, \dots, \tau_n$ - уже построенные термы сортов $A_1, A_2, \dots, A_n$ соответственно, то выражение $\mathbf{f}(\tau_1, \tau_2, \dots, \tau_n)$ есть терм сорта В.

B) Построение **атомарной формулы (атома)**. Атомы составляются из термов, **предикатных символов** и символа «=» (равенство) по следующим правилам.

П3. Если сигнатура ∑ содержит выражение

$$\mathbf{p: A_1 x A_2 x \dots x A_n \rightarrow T}$$

и $\tau_1, \tau_2, \dots, \tau_n$ - термы сортов $A_1, A_2, \dots, A_n$ соответственно, то выражение

$\mathbf{p}(\tau_1, \tau_2, \dots, \tau_n)$ есть атом.

П4. Если τ и σ - термы одинакового сорта, то выражение $\tau = \sigma$ есть атом.

C) Построение ППФ. ППФ составляется из атомов с использованием логических связок $(\neg, \wedge, \vee, \rightarrow)$ и кванторов $(\exists, \forall)$ по следующим правилам.

П5. Каждый атом есть ППФ. Всякая входящая ваттом переменная является **свободной** в этой формуле.

П6. Если α и β - уже построенные ППФ, то выражения

$\neg\alpha, (\alpha \wedge \beta), (\alpha \vee \beta), (\alpha \rightarrow \beta)$ - тоже ППФ.

Всякая переменная, входящая свободно в α и/или β, входит свободно и в новые формулы.

П7. Если α - уже построенная формула, в которую свободно входит переменная **x,** значения которой принадлежат сорту А, то выражения $(\exists x \in A)\alpha$ и $(\forall x \in A)\alpha$ также являются ППФ. В новых формулах переменная **x** считается **связанной**.

Назовем **замкнутой** ППФ (далее, просто **формулу**), не содержащую свободных переменных.
Рассмотрим, как формально представляются в языке многосортного исчисления предикатов следующие знания:

Знание 1. “Занятия по каждой дисциплине, запланированной в учебном процессе, должны быть проведены в одном из семестров”. Соответствующая формула может иметь вид:

(∀x∈ Дисциплина)
[((ном_семестра(проводится(x))=1) ∨((ном_семестра(проводится(x))=2) ∨
∨ ((ном_семестра(проводится(x))=3) ∨((ном_семестра(проводится(x))=4) ∨
∨ ((ном_семестра(проводится(x))=5) ∨((ном_семестра(проводится(x))=6) ∨
∨ ((ном_семестра(проводится(x))=7) ∨((ном_семестра(проводится(x))=8)]

Знание 2. “Дата начала семестра должна раньше даты его окончания”
Соответствующая формула может иметь вид:

(∀x∈ Семестр) [дата_нач(x) ≤ дата_окончан(x)]

2.3.3. Интерпретация сигнатуры

Итак, сигнатура задает структурные, функциональные и, частично, логические связи между понятиями ПО. Большая же часть логических связей задается формулами. Таким образом, сигнатура и формулы задают разные типы знаний о ПО.
Очевидно, формулы должны адекватно отображать соответствующие знания, то есть быть **истинными**. Каковы же необходимые и достаточные условия для определения истинности или ложности формул?
Ясно, что **необходимым** условием для определения истинности формулы является ее **замкнутость**. Это условие означает **связность** всех переменных, входящих в формулу, то есть определение для каждой переменной ее **типа** (значения констант какого сорта принимает переменная. См. правило 7 построения ППФ). Например, об истинности формулы

x<y (*)

ничего вообще сказать нельзя.
Замкнем обе свободные переменные **x** и **y** :

(∀x∈ Группа) (∃y∈ Аудитория) [размер(x) < вместимость(y)], (**)

Где:
размер(Группа) – функция, значением которой является количество студентов в группе (потоке),
вместимость(Аудитория) - функция, значением которой является количество посадочных мест в аудитории.
Но даже и при замкнутости судить об истинности формулы (**), не имея списка численности всех студенческих групп (потоков), в которых запланированы занятия, и сведений о вместимости всех аудиторий, невозможно. Необходимо определение **множеств значений**, которые могут принимать переменные данных сортов. Такие множества значений задаются с помощью определения **интерпретирующих структур**.
Замечание. Очевидно, что истинность формулы (**) априори очевидна, иначе учебный процесс не может быть полностью организован.
Обозначим через **Ω** множество замкнутых формул, построенных для ПО. Эти формулы должны быть истинны в той формальной (интерпретирующей) структуре, которая представляет ПО (если это представление ей адекватно).
Структура, интерпретирующая заданную сигнатуру ∑, строится следующим образом:

1. Указывается непустое множество **U**, называемое **универсумом**.
2. Для каждого имени сорта **A** из сигнатуры ∑ указывается некоторое подмножество универсума, которое также обозначается через **A** (**A**⊆ **U**).
3. Для каждого выражения **f**: $\mathbf{A_1 \times A_2 \times \ldots \times A_n \rightarrow B}$ из сигнатуры указывается функция, отображающая множество $\mathbf{A_1 \times A_2 \times \ldots \times A_n}$ во множество **B**; эта функция также обозначается **f**.
4. Для каждого выражения **p**: $\mathbf{A_1 \times A_2 \times \ldots \times A_n \rightarrow B}$ из сигнатуры указывается предикат, отображающий множество $\mathbf{A_1 \times A_2 \times \ldots \times A_n}$ во множество **T**; этот предикат также обозначается **p**.

Интерпретирующая структура для сигнатуры, описывающей фрагмент учебного процесса, может быть построена следующим образом.

U={$ст_i$, ном_зач$_j$, фио$_k$, гр$_l$, дисц$_m$ | i, j, k, l, m∈N, i, j, k, l, m, n≥1} ∪ {код_студ, ном_зач, фио, ном_гр, наим_дисц} ∪ N, где **N=0, 1,…**

Студент = {ст1, ст2, ст3, ст4, ст5};

Ном_Зач={ном_зач1, ном_зач2, ном_зач3, ном_зач4, ном_зач5}

ФИО = {фио1, фио2, фио3, фио4, фио5};

Группа = {гр1, гр2};

Дисциплина = {дисц1, дисц2, дисц3};

Оценка_Экзамен = {3, 4, 5}.

Оценка_Зачет = {“зачтено”, “не зачтено”}

Значения функций приведены в следующих таблицах.

Таблица 1

Студент	код_студ
студ1	01
студ2	02
студ3	03
…	…

Таблица 2

Ном_Зач	ном_зач
ном_зач1	01
ном_зач2	02
ном_зач3	03
ном_зач4	04
ном_зач5	05

Таблица3

ФИО	фио
фио1	Иванов И.И.
фио2	Петров П.П.
фио3	Котов Н.Н.
фио4	Ершов В.В.
фио5	Васин Д.Д.

Таблица 4

Группа	ном_гр
гр1	100
гр2	101

Таблица 5

Дисциплина	наим_дисц
дисц1	Информатика
дисц2	Математический анализ
дисц3	История

Таблица 6

Студент	Ном_Зач	ФИО	Группа	Дисциплина	Оценка_Экзамен	Оценка_Зачет
ст1	ном_зач1	фио1	гр1	дисц1	4	NULL
ст1	ном_зач1	фио1	гр1	дисц2	NULL	зачтено
ст1	ном_зач1	фио1	гр1	дисц3	5	зачтено
…	…	…	…	…	…	…

Замечание. Таблица 6 отображает структурные функции (1) – (7). Константа NULL означает, что по данной дисциплине не предусмотрен зачет (экзамен).

Таким образом, если заданы произвольная сигнатура $\sum$ и интерпретирующая структура (далее просто **структура**) **M,** то каждая замкнутая формула α **истинна** или **ложна** в структуре **M.**

Формальное определение истинности или ложности замкнутой формулы α в структуре **M**

Определяется согласно индуктивным правилам П8 и П9.

Для формулирования этих правил расширим исходную сигнатуру $\sum$**,** присоединив к ней выражения **a:** $\rightarrow$**A** для каждого элемента **a** сорта **A** и каждого сорта **A** из $\sum$ (таблицы 1 ÷ 5). Другими словами, расширенная сигнатура $\sum$**'** включает все литеральные константы, обозначающие элементы тех подмножеств структуры **M**, которые интерпретируют сорта. Благодаря такому расширению множество всех замкнутых формул Ω**'** оказывается замкнутым относительно подстановок любых литеральных констант вместо свободных переменных.

Далее тот факт, что замкнутая формула α истинна в структуре **M** будем обозначать как **M** $\models\alpha$ **.**

П8. Для любых замкнутых формул $\alpha,\beta\in\Omega$**'** :

M $\models\neg\alpha$ тогда и только тогда, когда не имеет место **M** $\models\alpha$**.**

M $\models(\alpha\wedge\beta)$ тогда и только тогда, когда одновременно **M** $\models\alpha$ и **M** $\models\beta$

M $\models(\alpha\vee\beta)$ тогда и только тогда, когда **M** $\models\alpha$ или **M** $\models\beta$

M $\models(\alpha\rightarrow\beta)$ тогда и только тогда, когда не имеет места **M** $\models\alpha$ и/или имеет место **M** $\models\beta$.

П9. Для любой формулы $\alpha\in\Omega$**'** , содержащей одну свободную переменную **x,** что обозначается как $\alpha=\alpha$**[x]**, полагаем

А) **M** $\models(\exists x\in A)\alpha$ тогда и только тогда, когда **M** $\models\alpha$**[a]** для хотя бы одной константы **a** сорта **A** из $\sum$'.

В) **M** $\models(\forall x\in A)\alpha$ тогда и только тогда, когда **M** $\models\alpha$**[a]** для всех констант сорта **A** из $\sum$'.

Таким образом, базу индуктивных правил П8 и П9 составляют константные (то есть, не содержащие переменных) атомарные формулы.

Структура **M**, интерпретирующая сигнатуру $\sum$, называется **модельной структурой** для заданной совокупности замкнутых формул Ω **,** если все формулы из Ω истинны в **M**, то есть **M** $\models\alpha$ для всех $\alpha\in\Omega$**.**

Совокупность замкнутых формул может не иметь ни одной модельной структуры. Тогда она называется **логически невыполнимым** (или просто **невыполнимым, противоречивым**) множеством.

2.3.4. Логические языки моделей данных и язык многосортной логики

В теории баз данных под **моделью данных** понимают формализм, предназначенный для описания структур данных, зависимостей между ними, а также ограничений целостности, то есть некоторых условий, которым должны удовлетворять допустимые базы данных.

Описание, выполненное в таком формализме, называется **схемой**. Допустимая база данных, то есть конечная совокупность данных, удовлетворяющих схеме, называется **экземпляром схемы**.

В указанной терминологии язык многосортного исчисления предикатов следует считать моделью данных. В самом деле, $\sum\cup\Omega$ можно рассматривать как схему, причем сигнатура $\sum$ описывает структуру данных, а множество замкнутых формул Ω - зависимости и ограничения целостности. Конечную модельную структуру **M** для Ω можно считать экземпляром схемы $\sum\cup\Omega$**.**

Очевидно, что реляционная модель данных - это формализм, непосредственно соответствующий простому фрагменту языка многосортного исчисления.

Пример. Набор экземпляров реляционных схем для данных фрагмента ПО учебного процесса может выглядеть следующим образом.

Студент

Код_ студента	Ном_Зач	ФИО	Группа	Дисциплина	Оценка_Экзамен	Оценка_Зачет
1	01	фио1	гр1	дисц1	4	NULL
1	01	фио1	гр1	дисц2	5	зачтено
1	01	фио1	гр1	дисц3	NULL	зачтено
…	…	…	…	…	…	…

ФИО

Код_фио	фио
фио1	Иванов И.И.
фио2	Петров П.П.
фио3	Котов Н.Н.
фио4	Ершов В.В.
фио5	Васин Д.Д.

Группа

Код_группы	ном_гр
гр1	100
гр2	101

Дисциплина

Код_дисц	Наим_дисц	Зачет	Экзамен	Ном_семестр
дисц1	Информатика	0	1	1
дисц2	Математический анализ	1	1	1
дисц3	История	1	0	1

Семестр

Номер_семестра	Дата_начала	Дата_окончания
1	1.09.2011	1.02.2012
…	…	…

Замечание. Приведенный пример реляционных отношений является чисто демонстрационным.

Реляционная база данных **R** включает пять реляционных **отношений** (**Студент, Ном_Зач, ФИО, Группа, Дисциплина**). Связь отношений с атрибутами указывается при помощи следующих структурных предложений:

- ✓ **Студент (Код_студента, Ном_Зач, ФИО, Группа, Дисциплина, Оценка_Экзамен, Оценка_Зачет),**
- ✓ **ФИО (Код_фио, фио),**
- ✓ **Группа (Код_группы, ном_гр),**
- ✓ **Дисциплина(Код_дисц, Наим_дисц, Зачет, Экзамен, Ном_семестр),**
- ✓ **Семестр(Номер_семестра, Дата_начала, Дата_окончания).**

Окончательно структурная часть **R'** реляционной схемы получается после определения для структурных предложений описания **доменов** (областей значений атрибутов):
dom(**ФИО**) = {“Иванов И.И.”,”Петров П.П.”, “Котов Н.Н.”, … }- символьные строки, представляющие ФИО;
dom(**Группа**) = {”100”, “101”, …} – символьные строки, представляющие целые числа;
dom(**Дисциплина**) = {“Информатика”, “Математический анализ”, “История”, …} – символьные строки, представляющие названия предметов;
dom(**Дата_начала**) = **dom**(**Дата_окончания**) = D – дата,

dom(**Номер_семестра**) = {1, 2, 3, 4, 5, 6, 7, 8}.
Сигнатура $\sum_1$ получается из **R'** трансляцией, при которой отношения и атрибуты реляционной схемы переходят в сорта сигнатуры, а каждая пара **<отношение-атрибут>** определяет одноместную функцию, отображающую элементы сорта, соответствующего отношению, в элементы сорта, соответствующего атрибуту.
Предложения, задающие **ключи** и **функциональные зависимости** относятся к логической части **R″** реляционной схемы **R**. Для нашего примера предложения, задающие ключи, будут иметь вид:

- ✓ (Код_студ, Дисциплина) **ключ в** Студент,
- ✓ Код_фио **ключ в** ФИО,
- ✓ Код_группы **ключ в** Группа,
- ✓ (Код_дисц, Ном_семестра) **ключ в** Дисциплина,
- ✓ Номер_семестра **ключ в** Семестр.

Эти предложения на языке многосортной логики имеют вид (для первого ключа):
(∀x,y∈ Студент).
[((Код_студ(x)=Код_студ(y)) ∧(Дисциплина(x)= Дисциплина(y))) → x=y]

Функциональная зависимость - это связь между набором атрибутов отношения и отдельным атрибутом этого отношения, определяющая ограничение на экземпляры схемы: в каждом кортеже значения атрибутов из набора **однозначно** определяют значение другого атрибута. Функциональная зависимость выражается предложениями вида:
(**A1, A2, … ,Ap**) **определяют B в R**, где
A1, A2, … ,Ap, B - атрибуты отношения **R**.
Например: **Код_студ определяет Код_группы в Студент**
Функциональные зависимости также можно отнести к логической части **R″** реляционной схемы. Они выражаются простыми формулами многосортной логики:
(∀x,y∈ Студент).
[(Код_студ(x) = Код_студ(y)) → (Код_группы(x) = Код_группы(y))]
Предложения реляционной схемы **R**, отличные от структурных предложений, ключей и функциональных зависимостей, выражают ограничения целостности. Например, знание Знание2 следует отнести к ограничению целостности (дата начала семестра должна быть раннее даты его окончания).
На основе исчисления предикатов можно строить и языки запросов к реляционной базе данных, пример - **SQL**.
Пример. Запрос: "Выдать ФИО студентов, получивших в первом семестре оценку «отлично» по информатике."
Этот запрос можно сформулировать как нахождение множества
{(x.ФИО) | (∃x∈ Студент) (∃y∈ Дисциплина). [(изучает(x)=y) ∧ (Оценка_экзамен(y)=5) ∧
∧ (Ном_семестр(y)=1]}
Аналогичный запрос на языке SQL выглядит следующим образом:

```
SELECT ФИО
FROM Студент, Дисциплина
WHERE ((Студент.Оценка_экзамен=5) AND
       (Студент.Дисциплина=Дисциплина.Код_дисц) AND
       (Дисциплина.Ном_семестр=1))
```

2.3.5. Логическое следствие и логический вывод

С помощью понятия модельной структуры уточняется понятие логического следствия, которое интуитивно является достаточно ясным.
Пусть **Ω** - произвольное множество замкнутых формул и **β** - произвольная замкнутая формула, записанные в сигнатуре $\sum$. Говорят, что **β** логически следует из **Ω**, и пишут **Ω|=β,** если **β** истинно во всякой модельной структуре для **Ω.** Иными словами, отношение логического

следствия **Ω|=β** имеет место тогда и только тогда, когда нет ни одной интерпретации, в которой все формулы из **Ω** были бы истинны, а **β** - ложно. Таким образом, символ «|=» имеет двоякое употребление:

- для обозначения отношения **M |= β** между структурой и формулой,
- для обозначения отношения **Ω|=β** между множеством формул и формулой.

Для исчисления предикатов основной является следующая проблема логического следствия: разработать методы и алгоритмы, позволяющие для произвольных **Ω** и **α** - множества замкнутых формул и замкнутой формулы, выяснить, имеет ли место отношение логического следствия **Ω|=α.**

Проблема логического следствия тесно связана с проблемой **общезначимости** логических формул.

Замечание. Замкнутая формула **α** называется **общезначимой** (или **тождественно истинной**), если **α** истинна во всякой структуре сигнатуры ∑.

Легко доказать, что соотношение $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}|=\beta$ имеет место тогда и только тогда, когда формула $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \rightarrow \beta$ **(!)** общезначима.

Действительно, указанное логическое следствие справедливо тогда и только тогда, когда нет ни одной структуры **M** такой, что $M|=\alpha_1$, $M|=\alpha_2$, …, $M|=\alpha_n$ и не имеет место $M|=\beta$, а это значит, что нет ни одной структуры такой, что не имеет места $M|=\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \rightarrow \beta$.

С другой стороны, соотношение $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}|=\beta$ справедливо тогда и только тогда, когда формула $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \neg \beta$ невыполнима, то есть не имеет модельных структур.

Остается только найти алгоритм определения общезначимости формулы (!), и будет найден общий механизм проверки логического следствия !

Однако, А.Черч доказал алгоритмическую неразрешимость проблемы общезначимости формул исчисления предикатов: нельзя построить алгоритм, который был бы применим к любой замкнутой формуле **α** в произвольной сигнатуре и который выяснял бы, является ли **α** общезначимой формулой или нет.

Замечание. Очевидно, что теорема Черча не запрещает существования алгоритмов общезначимости для отдельных сигнатур или классов сигнатур.

Примечание. Вкратце, рассуждения Черча заключаются в следующем.

Пусть формула $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \rightarrow \beta$ общезначима. Она истинна, если либо

1) $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$ всегда ложно. Другими словами, для всякой структуры **M** найдется хотя бы одна формула α_i ,не являющаяся истинной в **M**, то есть **не являющаяся общезначимой.**

2) β - общезначима.

Другими словами, для определения общезначимости произвольной замкнутой формулы в произвольной сигнатуре необходимо определить общезначимость произвольной замкнутой формулы (α_i или **β**) в произвольной сигнатуре!

2.3.6. Логическое следствие и выводимость

Теорема Геделя. В математической логике известна теорема Геделя, из которой следует, что можно построить **частичный**, то есть «зацикливающий» на некоторых предъявляемых ему формулах, алгоритм, область определения которого (то есть множество формул, на которых он дает результат) включает множество всех общезначимых формул. Теорема Геделя утверждает, что проблему общезначимости для исчисления предикатов можно решить с помощью системы логического вывода (дедукции) **гильбертовского типа**.

Система дедукции **гильбертовского типа** состоит **аксиом** и **правил вывода.**

Аксиомами служат некоторые априори заданные **общезначимые формулы**.

Правило вывода - это отношение $\mathbf{R}(\alpha_1, \alpha_2, \ldots, \alpha_n)$ между формулами α_i (i=1,…,n),

Обладающее свойством **самостоятельности:** если формулы $\alpha_1, \alpha_2, \ldots, \alpha_{n-1}$ истинны в какой-либо структуре **M** и имеет место $R(\alpha_1, \alpha_2, \ldots, \alpha_n)$, то формула α_n также истинна в **M**.

Вместо $R(\alpha_1, \alpha_2, \ldots, \alpha_n)$ обычно пишут $\alpha_1, \alpha_2, \ldots, \alpha_{n-1} \vdash \alpha_n$ и говорят, что формула α_n получена из формул $\alpha_1, \alpha_2, \ldots, \alpha_{n-1}$ применением к ним правила вывода **R**.

Выводом формулы **β** из множества формул **Ω** называется последовательность формул из **Ω** такая, что ее последняя формула совпадает с **β**, а каждая из остальных формул последовательности либо является аксиомой, либо получается из каких-либо предыдущих формул применением одного из правил вывода.

Если существует вывод формулы **β** из множества формул **Ω**, то говорят, что **β выводима** из **Ω,** что обозначается как $\Omega \vdash \beta$.

Система дедукций называется **полной**, если для любого множества формул **Ω** формула **β** выводима тогда и только тогда, когда **β** логически следует из **Ω**, то есть $\Omega \vdash \beta$ тогда и только тогда, когда $\Omega \models \beta$.

Таким образом, теорема Геделя устанавливает факт полноты **некоторой системы дедукции**.

Замечание. Кроме гильбертовых, существуют и другие системы дедукции - системы правил логического вывода.

Понятие формальной системы. В общем случае модели, основанные на логике предикатов, описываются **формальной системой L**, которая задается четверкой:

L=(T, S, A, P), где

T - множество **базовых элементов** (**терминальных** знаков: имена сортов, переменных, констант и т.п.) или **алфавит** формальной системы,

S - множество **синтаксических правил**, с помощью которых из элементов **T** можно строить **синтаксически правильные конструкции** (слова, предложения, высказывания, формулы и т.п.),

A - множество **аксиом**: синтаксически правильных тождественно истинных выражений, заданных априорно,

P - множество **правил вывода** (**семантических правил**, **продукций**), с помощью которых можно расширять множество **A**.

2.3.7. Достоинства и недостатки логических моделей представления знаний

Достоинства

Основным достоинством логических моделей является наличие единообразной формальной процедуры доказательства теорем (организация вывода) и, как следствие, возможность непосредственно запрограммировать механизм вывода синтаксически правильных высказываний.

Недостатки

1. Высокая степень единообразия влечет основной недостаток - сложность использования при построении доказательств эвристик, отражающих специфику конкретной предметной области.
2. Только с помощью правил, задающих синтаксис языка, нельзя установить истинность или ложность высказываний: синтаксически правильные высказывания могут быть семантически абсолютно бессмысленными.
3. Отсутствие средств структурирования используемых элементов, Без таких средств большая модель превращается в плохо обозримый конгломерат независимых фактов, трудно поддающихся анализу и обработке.
4. Недопустимость противоречий.

Логические модели представления и манипулирования знаниями были особенно популярными в 70-е годы, когда казалось, что с появлением специализированных языков программирования, предназначенных для решения логических задач, процедур логического вывода в исчислении предикатов, представленных средствами этих языков, будет достаточно для решения всех типов

задач в интеллектуальных системах. Однако, по мере расширения спектра интеллектуальных задач стало ясно, что говорить о доказательном выводе можно только в небольшом числе случаев, когда ПО, в которой решается задача, формально описана и полностью известна.
Однако большинство задач связано с областями, где знания принципиально неполны, неточны, некорректны и т.д.
При таких условиях речь может идти только о **правдоподобном** выводе, при котором окончательный результат получается лишь **с некоторой оценкой** уверенности в его истинности. Кроме того, специалисты, работающие в плохо формализуемых областях (психология, медицина), рассуждают совсем не так, как представители точных наук. Для них весомым аргументом в пользу принятия какого-либо решения могут быть мнения ряда признанных в этих областях авторитетов или, например, сходство доказываемого положения с другим, для которого решение уже известно (постановка диагноза).
Поэтому дальнейшее развитие логических моделей представления знаний пошло по пути работ в области индуктивных логик, логик «здравого смысла», логик веры, нечетких логик, немонотонных рассуждений и многих других логических систем, мало что общего имеющих с классической математической логикой.

2.4. Продукционные Модели Представления Знаний

2.4.1. Формальное описание

Продукционные модели наряду с фреймами являются наиболее популярными средствами представления знаний в ИС. Продукции, с одной стороны, близки к логическим моделям, что позволяет организовать на них эффективные процедуры вывода, а с другой стороны, более наглядно отображают знания, классические логические исчисления, что дает возможность изменять интерпретацию элементов продукции.
В общем виде под **продукцией** понимается выражение вида:

(i) ; Q ; P; A⇒B ; N, где

i - **идентификатор** продукции, с помощью которого осуществляется поиск продукции в базе знаний; в качестве имени может выступать некоторая **лексема**, отражающая суть продукции, например «покупка книги», или просто порядковый номер;

Q - элемент характеризует **сферу применения продукции**. Разделение знаний на отдельные сферы позволяет экономить время на поиск нужных знаний.

A⇒B - основной элемент продукции, ее **ядро**, часто называемое **правилом**.

Интерпретация ядра продукции может быть различной и зависит от того, что стоит слева и справа от знака секвенции ⇒. Обычное прочтение ядра продукции выглядит следующим образом:

ЕСЛИ A, ТО B;

более сложные конструкции ядра допускают в правой части альтернативный выбор, например,

ЕСЛИ A_1 ТО B_1, ИНАЧЕ B_2.

Секвенция может истолковываться в обычном логическом смысле как знак логического следования **B** из **A** (если **A** ложно, то о **B** ничего сказать нельзя). Однако возможны и другие интерпретации ядра продукций, например, **A** описывает некоторое условие, необходимое для совершение действия **B**.

P - **условие применимости** ядра продукции. Обычно **P** представляет собой логическое выражение (как правило, предикат). Когда **P** принимает значение **истина**, ядро продукции активизируется, в противном случае ядро продукции не может быть использовано. Например, если в продукции

“НАЛИЧИЕ ДЕНЕГ; ЕСЛИ ХОЧЕЩЬ КУПИТЬ ВЕЩЬ **X**, ТО ОПЛАТИ В КАССЕ ЕЕ СТОИМОСТЬ И ОТДАЙ ЧЕК ПРОДАВЦУ”

условие применимости ядра ложно (денег нет), применить ядро продукции невозможно.

N - элемент описывает **постусловия продукции**. Они актуализируются только в том случае, если ядро продукции реализовалось. Постусловия описывают действия и процедуры, которые

необходимо выполнить после реализации **B**. Например, после покупки некоторой вещи в магазине необходимо уменьшить количество товара данного типа в описи имеющихся товаров.

2.4.2. Классификация ядер продукции

Ядра продукций можно классифицировать по различным основаниям. Прежде всего все ядра делятся на два больших типа: **детерминированные** и **недетерминированные**. Секвенция $\Rightarrow$ в детерминированных ядрах реализуется с **необходимостью**, а в недетерминированных - **с возможностью**.

Детерминированные ядра могут быть **однозначными** и **альтернативными.** Во втором случае в правой части ядра указываются альтернативные возможности выбора, которые оцениваются специальными **весами выбора**. В качестве таких весов могут использоваться оценки разных типов:

вероятностные (ЕСЛИ А, ТО С ВЕРОЯТНОСТЬЮ 0.6 НАДО ДЕЛАТЬ B_1,А С ВЕРОЯТНОСТЬЮ 0.4 B_2);

лингвистические (ЕСЛИ А, ТО ЧАЩЕ НАДО ДЕЛАТЬ B_1, РЕЖЕ B_2);

и т.п.

Возможность реализации в **недетерминированных** ядрах определяется **оценками реализации** ядра: **ЕСЛИ А, ТО ВОЗМОЖНО В**. Оценки могут быть:

вероятностные (ЕСЛИ А, ТО С ВЕРОЯТНОСТЬЮ 0.8 НАДО ДЕЛАТЬ В);

лингвистические (ЕСЛИ А, ТО С БОЛЬШОЙ ДОЛЕЙ УВЕРЕННОСТИ В);

и т.п.

Особым типом являются прогнозирующие продукции, в которых описываются последствия, ожидаемые при реализации **А**:

ЕСЛИ А, ТО С ВЕРОЯТНОСТЬЮ p МОЖНО ОЖИДАТЬ В.

Классификацию ядер продукций можно провести, опираясь на типовую схему ИС (рис.11.).

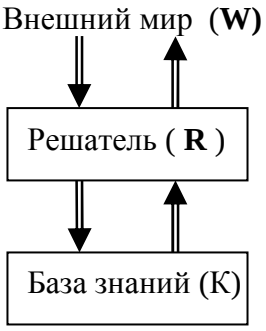


Рис.11. Классификация ядер продукции

В этой схеме используются продукции следующих типов:

1. **Aw $\Rightarrow$ BR.** В левой части продукции указана информация, поступившая из внешнего мира, а в правой - сведения о вытекающих из этой информации изменениях в рассуждающей системе. Очевидно, что эти изменения сказываются на ходе рассуждений.

   Пример. Вечером договариваемся о воскресном выходе на природу. Утром узнаем из сводки погоды, что ожидается дождь **(Aw).** В ответ меняется весь ход рассуждений о предпочтительном варианте отдыха **(BR)**. Сама продукция могла бы выглядеть следующим образом:

   “ЕСЛИ на улице идет дождь или гроза или они ожидаются в течение дня ТО вместо прогулки лучше пойти в кино или музей”

   В качестве **Aw** могут выступать: факты, имеющие место в **W**, сообщения о **W,** прямое воздействие из внешнего мира на рассуждающую систему. Вне зависимости от вида **Aw**, в правой части продукции стоят некоторые операторы, меняющие ход рассуждений.

2. **Aw ⇒ Bk.** Такие продукции отражают ситуацию передачи некоторого сообщения из внешнего мира для запоминания в базе знаний.
   Пример. Приказание командира разведки: ”Все, что увидишь интересного в окрестности переправы, запомни, а потом передай через связного”
   При работе с продукциями такого типа решатель выступает в роли отделения связи, передающего сообщение от одного абонента другому.
   Замечание. Решатель может «вскрыть» корреспонденцию, воспользовавшись информацией для своих целей.
3. **Ak ⇒ Bw.** Решатель выступает в качестве отделения связи при выдаче сообщения из базы знаний во внешний мир.
   Пример. Обнаружение в базе знаний противоречивой информации. Здесь **Ak** - факт наличия противоречия, **Bw** - действия, принимаемые решателем в связи с обнаружением данного факта.
4. **AR ⇒ Bk.** Некоторый факт, полученный решателем, передается на хранение в базу знаний.
5. **Ak ⇒ BR.** Описание обмена информацией при работе решателя. Некоторая информация выбирается из базы знаний и передается для обработки в решатель.
6. **Aw⇒ Bw.** Продукции непосредственного отклика.
   Aw - описание некоторой наблюдаемой ситуации во внешнем мире или воздействие внешнего мира на решатель.
   Bw - Описание действия, которое поступает от системы в окружающий мир.
   Решатель в этих случаях не успевает сработать, он лишь транслирует информацию от адресата **Aw** к адресату **Bw** (эффект «отдергивания руки»).
7. **AR⇒ Bw.** Описание воздействий во внешний мир, которые порождает результат работы решателя.
   Пример. “Подумай, прежде чем сделать” - совет воспользоваться продукцией данного типа, а не продукцией непосредственного отклика.
8. **AR⇒ BR**. Внутренние продукции рассуждающей системы. Описывают промежуточные шаги процесса вывода и не влияют непосредственно на содержимое Базы знаний и состояние внешнего мира. Эти продукции описывают единичные шаги многошаговых процессов рассуждений.
9. **Ak ⇒ Bk.** Описание процедур преобразования знаний в базе знаний:
   - обобщение знаний;
   - получение новых знаний из ранее известных с помощью логического вывода;
   - установление закономерностей между знаниями на основании обработки сведений о единичных фактах; хранящихся в базе знаний;
   - и т.п.

   Решатель в этом случае используется лишь в качестве инструмента, с помощью которого происходит изменение состояния базы знаний.

Очевидно, что возможны и другие интерпретации. Например, продукции типа **Aw ⇒ Bk** можно трактовать как способ описания шагов общения между пользователем и системой в диалоговом режиме. Тогда **Aw** будет трактоваться как вопрос пользователя, а **Bk** - ответ системы. При смене инициаторов диалога рассматриваются продукции типа **Ak ⇒ Bw.**

2.4.3. Структура продукционной системы и стратегии вывода

Структурно продукционная система состоит из трех основных компонентов (рис.12.).

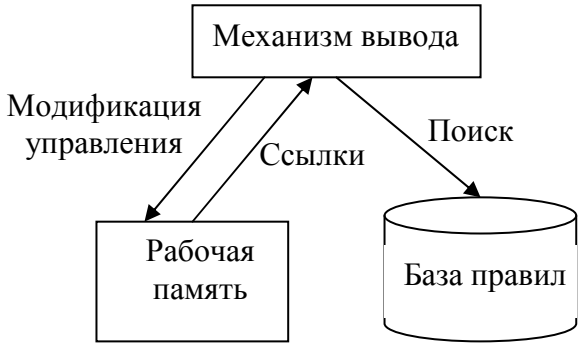


Рис. 12. Структурная схема продукционной системы

База правил - набор правил, используемый как база знаний.
Рабочая память (память для **кратковременного хранения**) - хранит предпосылки, касающиеся конкретных задач ПО, и результаты выводов, полученных на их основании.
Механизм логического вывода - использует правила в соответствии с содержимым рабочей памяти.
Существует две общие стратегии логического вывода: **прямой** и **обратный**.
Стратегия прямого вывода. В системах с прямым выводом по известным фактам отыскивается заключение, которое из этих фактов следует. Если такое заключение удается найти, оно заносится в рабочую память (РП). Прямой вывод часто называют **выводом, управляемым данными.**

Пример. Пусть данные, записанные в РП, представляют собой **образцы** в виде наборов символов, например, следующие текстовые данные (факты):

- “намерение - отдых”
- “место отдыха - горы”

Правила, накапливаемые в базе знаний, отражают содержимое РП. В их условной части находятся либо одиночные образцы, либо образцы, соединенные предлогами **И** и **ИЛИ,** а в правой части - образцы, которые должны быть зарегистрированы в РП (то есть, должны быть занесены в РП, если их там нет после выполнения соответствующего правила).
Пусть в базе знаний хранятся следующие правила:
Правило 1. **ЕСЛИ** (“намерение - отдых” **И** “дорога ухабистая”) **ТО** “использовать джип”
Правило 2. **ЕСЛИ** “место отдыха - горы” **ТО** “дорога ухабистая”
Общая схема работы механизма прямого вывода.

1. Механизм вывода сопоставляет образцы из условных частей правил, хранящихся в БЗ. С образцами, занесенными в РП.
   Если для какого-либо правила в РП имеются все образцы, необходимые для того, чтобы условная часть этого правила принимала значение **истина**, правило выполняется и его правая часть (факт) заносится в РП, если его еще там не было.
   В нашем примере выполниться может только Правило 2. После его выполнения содержимое РП принимает вид:
   - “намерение - отдых”
   - “место отдыха - горы”
   - “дорога ухабистая
2. Сработавшее на очередном цикле работы механизма вывода правило в следующих циклах блокируется и поиск подходящего для выполнения правила ведется среди оставшихся правил. В нашем примере для анализа на применимость остается одно Правило 1. Для его реализации в РП есть все данные, поэтому оно срабатывает и содержимое РП принимает следующий вид:
   - “намерение - отдых”
   - “место отдыха - горы”

- “дорога ухабистая
- “использовать джип”

3. Очевидно, что в общем случае механизм заканчивает свою работу в случае, когда в БЗ нет ни одного применимого для текущего состояния РП правила, либо когда все правила БЗ выполнились, что имеет место в нашем примере.

Стратегия обратного вывода. В системах с **обратным выводом** в начале выдвигается некоторая **гипотеза** (**цель**) Если эта цель сразу достигнута быть не может, ставится промежуточная цель, детализирующая первую и являющаяся по отношению к ней подцелью, то есть механизм вывода в процессе работы как бы возвращается назад, переходя от цели к фактам и пытаясь найти среди них те, которые позволяют достичь цель. Достигнутые подцели позволяют подтвердить гипотезы более высоких порядков вплоть до достижения первоначальной цели.

Пример.

1. Пусть исходная цель - “использовать джип”.
2. В условиях предыдущего примера для достижения этой цели достаточно на основании Правила 1 подтвердить факт “дорога ухабистая”.
3. Этот факт в РП отсутствует, поэтому становится новой целью (подцелью).
4. Исследуется возможность применения Правила 2. Правило применимо, подцель “дорога ухабистая” достигнута, этот факт заносится в РП.
5. В РП достаточно фактов для достижения первоначальной цели.

В случае обратного вывода условия остановки системы очевидны: либо достигается первоначальная цель, либо исчерпываются правила, применимые для достижения цели в ходе вывода.

Обратный вывод часто называют **выводом, управляемым целями.** Такая стратегия эффективна, когда цели заранее известны и их сравнительно немного.

Прямой вывод чаще применяют в системах **диагностики**, а обратный - в системах **прогнозирования**.

На сочетании ограниченного прямого и обратного выводов основывается часто применяемый на практике метод **циклического вывода**.

2.4.4. Продукционные модели и Модули, Управляемые Образцами

Общее описание и классификация

Традиционный способ программирования удобен в тех случаях, когда последовательность обработки мало зависит от обрабатываемых данных, то есть ветвление является исключением, а не нормой. В противном случае программу лучше рассматривать как совокупность независимых модулей, **управляемых образцами**. На каждом шаге работы такая программа анализирует текущую ситуацию и определяет по анализу образцов, какой модуль подходит для обработки этой ситуации.

Каждый управляемый образцом модуль (УОМ) состоит из механизмов исследования и модификации одной или нескольких структур данных. Диапазон УОМ может колебаться в широких пределах от простого продукционного правила до процедуры произвольной степени сложности, вызываемой по образцу. Каждый УОМ на очередном шаге работы анализирует данные рабочей памяти, проверяя наличие структур, которые сопоставляются с его образцом.

Системы, построенные на основе управляемых образцами модулей, называют **системами вывода, управляемыми образцами.** Функции управления в таких системах осуществляет **интерпретатор**, выполняющий роль **решателя**.

С точки зрения представления знаний подход, использующий УОМ, характеризуется следующими особенностями:

1. Разделение знаний на **постоянные**, хранимые в базе знаний, и **временные**, располагающиеся в рабочей памяти.

2. **Структурная независимость модулей.** Это свойство облегчает модификацию и совершенствование системы, что чрезвычайно важно для **экспертных систем,** постоянно модифицирующих свои знания.
3. Отделение схемы управления от модулей, несущих знания о предметной области. Данное свойство позволяет применять разные системы управления.
4. **Классификация систем, управляемых образцами**
5. Системы, управляемые образцами, классифицируются в соответствии с ограничениями, накладываемыми на модули. Приведенная ниже схема может рассматриваться как классификация УОМ (а также систем, управляемых образцами) (рис.13).

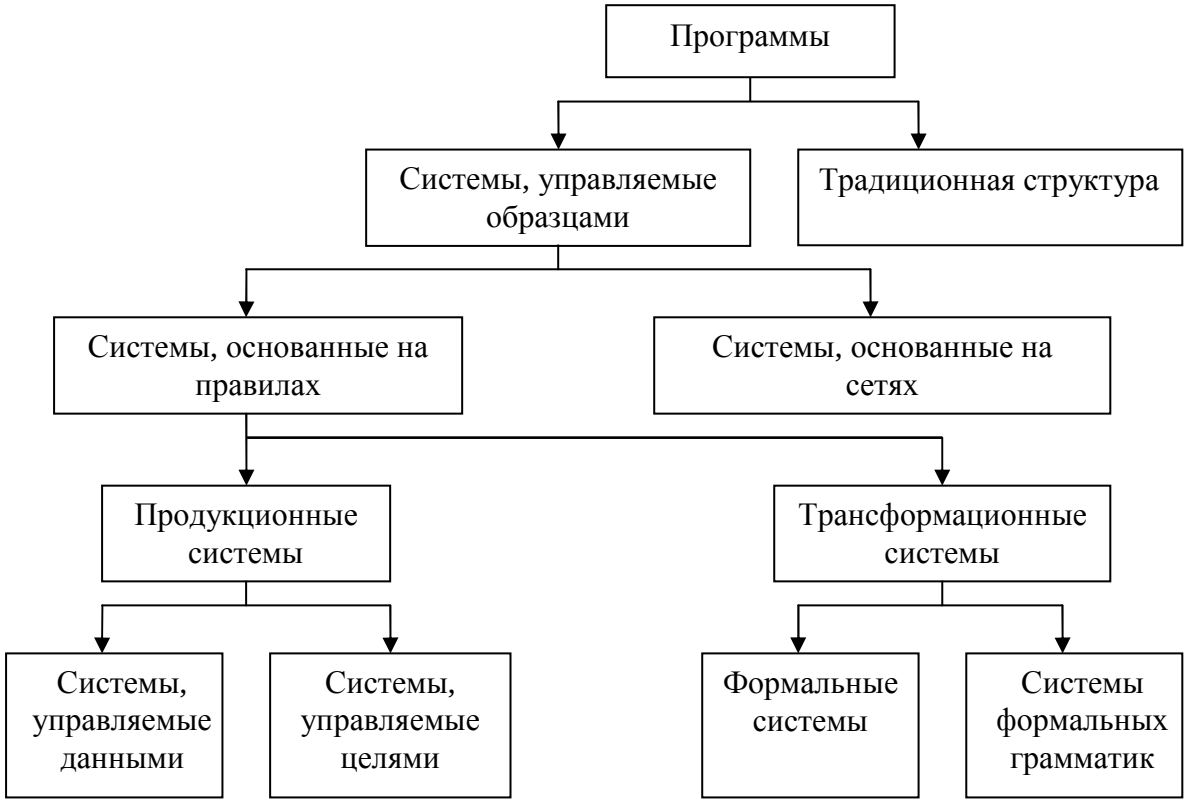


Рис.13 Классификация систем, управляемых УОМ

1. Если системы, управляемые образцами, состоят из модулей, локализованных в вершинах сети, которые активизируются сигналами, приходящими по дугам, такие системы называются **системами, основанными на сетях**.
2. Большинство систем, управляемых образцами, удовлетворяют следующему ограничению: все исследования по сопоставлению образцом модулей и данных в рабочей памяти предшествуют любым действиям по обработке (модификации) данных. Таким образом, УОМ разделяется на две части: **предусловие**, исследующее данные, и **действие**, модифицирующее данные. Модули, имеющие такое деление, называются **правилами**, а системы, использующие такие правила **системами, основанными на правилах.**
3. Системы, основанные на правилах, разделяются по **видам правил** на **продукционные** и **трансформационные.**
   Продукционные системы образованы из правил, в которых сопоставление и управление (планирование) являются **явными функциями** системы, зафиксированными в интерпретаторе.

В трансформационных системах функции управления могут быть явно не выражены. Примерами трансформационных систем являются системы формальных грамматик и формальные системы.

4. Продукционные системы могут быть разделены на системы, **управляемые данными** (предусловиями правил) и системы, **управляемые целями (**действиями правил). Традиционно под продукционными системами понимают только системы, использующие вывод, управляемый данными (прямой вывод).

Обычно предусловие задается в виде логической комбинации утверждений о данных рабочей памяти, а действием является некоторая операция по модификации содержимого рабочей памяти. Сложность действий может колебаться от простого оператора присваивания до выполнения процедуры произвольной сложности.

2.4.5. Достоинства и недостатки продукционных моделей

Достоинства. Популярность продукционных моделей определяется несколькими факторами:

1. Подавляющая часть человеческих знаний может быть представлена в виде продукций.
2. Системы продукций являются модульными. За небольшим исключением удаление или добавление продукций в базу знаний не приводит к изменениям в остальных продукциях.
3. При необходимости системы продукций могут реализовывать любые алгоритмы и, следовательно, способны отражать любое процедурное знание, доступное ЭВМ.
4. Наличие в продукциях указателей на сферу применения продукции позволяет эффективно организовывать память, сократив время поиска в ней необходимой информации. Классификация сфер может быть многоуровневой, что еще более повышает эффективность поиска знаний, так как позволяет наследовать информацию в базе знаний.
5. При объединении систем продукций и сетевых представлений получаются средства, обладающие большой вычислительной мощностью. В таких моделях декларативные знания описываются в сетевой компоненте модели, а процедурные знания - в продукционном. В этом случае говорят о работе продукционной системы над семантической сетью.
6. Естественный параллелизм в системе продукций, асинхронность их реализации делают продукционные системы удобной моделью вычислений для ЭВМ параллельных архитектур.

Недостатки. Продукционные системы имеют, по крайней мере, два недостатка:

1. При большом количестве продукций становится сложной проверка непротиворечивости системы продукций (эту проверку приходится делать при каждом добавлении новой продукции).
2. Из-за присущей системе недетерминированности (неоднозначного выбора выполняемой продукции из фронта активных продукций) возникают принципиальные трудности при проверке корректности работы системы.

Основной причиной недостатков продукционных моделей является нехватка строгой теоретической основы. При задании модели предметной области в виде совокупности продукций нельзя быть уверенным в ее полноте и непротиворечивости. Причина неудач создания теории кроется в расплывчатости понятия продукции, в той интерпретации, которая приписывается ядру, а также в различных способах управления системой продукций. Переход к алгоритмической схеме мало, что дает, поскольку в этом случае теряется основное свойство продукций - их модульность и вместе с ним асинхронность и параллельность выполнения продукций в системе.

Другая проблема практического применения продукционных моделей - переход от статических систем к динамическим, меняющим состав продукций в системе или перестраивающих алгоритм управления выбором продукций из фронта готовых продукций в зависимости от

текущих ситуаций. Такие системы продукций называются **адаптивными** или системами продукций **реального времени**.

3. ЭКСПЕРТНЫЕ СИСТЕМЫ

3.1. Назначение и особенности

Классифицируем знания специалистов как: **формализованные** (точные) и **неформализованные** (неточные).

Неточные знания характеризуются **неконкретностью**, **субъективностью**, **приблизительностью**. Знания этого рода являются результатом обобщения многолетнего опыта работы и интуиции специалистов и обычно представляют собой многообразие эмпирических (эвристических) приемов и правил.

Назовем **формализованными** задачи, основанные на **точных** знаниях, и **неформализованными** - на **неточных**.

К **неформализованным** можно отнести задачи, обладающие одной или несколькими следующими **характеристиками**:

- задачи не могут быть заданы в числовой форме;
- цели не могут быть выражены в терминах точно определенной целевой функции;
- не существует алгоритмического решения задачи;
- алгоритмическое решение существует, но его нельзя использовать из-за ограничения ресурсов вычислительной системы.

Неформализованные задачи обычно обладают следующими **особенностями**:

- ошибочность, неоднозначность, неполнота и противоречивость данных;
- ошибочность, неоднозначность, неполнота и противоречивость знаний о ПО;
- динамически изменяемые данные и знания;
- большой размер пространства состояний, не допускающий простой перебор при поиске решения.

Особенности экспертных систем. Определение экспертной системы

А) Отличие систем искусственного интеллекта (ИС) и, в частности, экспертных систем (ЭС) от систем обработки данных (СОД) состоит в том, что в этих системах используются:

- символьный, а не числовой способ представления данных;
- символьный вывод (поиск решения);
- эвристический поиск решения, а не готовое решение (алгоритм).

В) Специфика ЭС по сравнению с другими типами ИС состоит в следующем:

1. ЭС применяется для решения только практически трудных задач.
2. по качеству и эффективности решение ЭС не уступают решениям человека-эксперта.
3. решения ЭС обладают “прозрачностью”, то есть могут быть объяснены пользователю на качественном уровне (в отличие от решений, полученных числовыми, в особенности, статистическими методами). Это качество ЭС обеспечивается их способностью рассуждать о своих знаниях и умозаключениях.
4. Знания, позволяющие ЭС получать качественные и эффективные решения задач, являются в основном эвристическими, экспериментальными, неопределенными, правдоподобными. Причина этого - решаемые задачи являются не- или слабо формализованными. ЭС способны пополнять свои знания в ходе диалога с экспертом.
5. Обеспечение “дружественного”, как правило, ЕЯ-интерфейса с пользователем.
6. Мощность ЭС обусловлена в первую очередь мощностью базы знаний, и только во вторую - используемыми методами (процедурами) вывода. То есть важнее иметь разнообразные специальные знания, а не общие процедуры вывода.

Таким образом, ЭС называют машинную систему, достигающую высокого уровня работы в таких областях, в которых человеку необходимо потратить несколько лет на образование и приобретение опыта.

ЭС - это специализированная компьютерная система, способная к накоплению и обобщению эмпирического опыта высококвалифицированных экспертов в заданной ПО с последующей

консультацией рядовых специалистов в их повседневной деятельности, включая объяснение логики получения решения.
ЭС в процессе работы способна оперативно находить ответы на вопросы о том, **что**, **где**, **когда** и **почему** случилось, как поступить в данном)текущем) состоянии, насколько эффективны будут предпринятые действия, каков будет результат текущего действия, как поступить, если ожидания не оправдаются и т.п.

3.2. Структура ЭС

Общая структура ЭС приведена на рис.14.

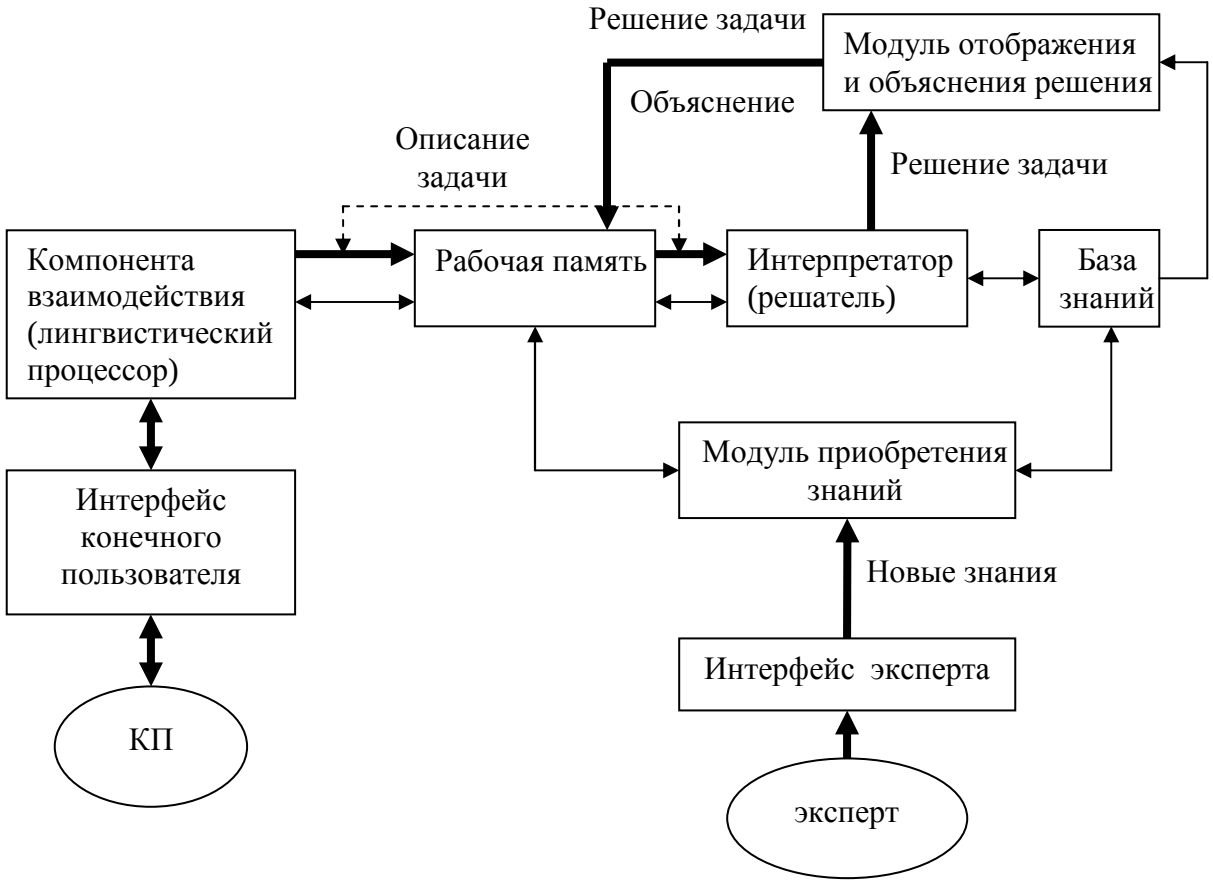


Рис.14. Обобщенная схема ЭС

1. Взаимодействие пользователя с ЭС осуществляется через **интерфейс конечного пользователя** на проблемно-ориентированном языке непроцедурного типа. **Лингвистический процессор** производит преобразование (трансляцию) предложений естественного языка (или другого проблемно-ориентированного языка непроцедурного типа) на внутренний язык представления знаний данной ЭС.
2. Описание задачи (запроса) пользователя на внутреннем языке представления знаний поступает в **подсистему логического вывода** (**Интерпретатор**), которая, используя содержимое БЗ, генерирует рекомендации по решению искомой задачи. Основу базы знаний ЭС составляют факты и правила. В подсистеме логического вывода реализуется некоторая стратегия выбора соответствующего правила из БЗ, тесно связанная со способом представления знаний в ЭС и характером решаемых задач.
3. С помощью подсистемы **отображения и объяснения решения** происходит отображение промежуточных и окончательных решений и объяснение пользователю действий системы. Как правило, система отвечает на вопросы “как” и ”почему”, то есть на вопросы типа ”как достигнуто то или иное заключение”, ”почему оно было достигнуто”, ”почему отброшены другие альтернативы”.

Считается, что наличие подсистемы объяснения является то важной особенностью ЭС, которая обеспечивает необходимый уровень доверия пользователя к системе. Более того, неявно предполагается, что если метод рассуждения, реализованный в ЭС, не может быть объяснен пользователю, то он должен быть признан неудовлетворительным.
Эта характеристика отличает ЭС от прочих типов систем ИИ, где при функционировании **Решателя** важно само получение решения, и не столь существенен процесс объяснения получения этого решения.

4. Функция подсистемы **приобретения знаний** состоит в поддержке процесса извлечения знаний о соответствующей узкоспециализированной ПО. Как правило, эти знания, носящие эмпирический характер, плохо формализованы и отсутствуют в специальной литературе (учебниках, монографиях, инструкциях и т.п.). Такие знания приобретаются экспертом в результате длительного опыта.
   Замечание. При проектировании ЭС процесс передачи таких знаний от эксперта системе является наиболее узким местом.
5. **База знаний** хранит множество продукций (в общем случае правил).
6. **Рабочая память** - это хранилище данных (фактов), в роли которого обычно выступает база данных.

Очевидно, что ключевая роль в накоплении знаний в базе знаний системы принадлежит **инженеру знаний.** В его задачу входит выявление знаний экспертов-специалистов, их четкая формулировка (формализация), компьютеризация и введение в ЭВМ.
ЭС работает в двух режимах: режим **приобретения знаний** и режим **решения задач** (собственно решений, консультаций).
Архитектура реальных ЭС различается в первую очередь по следующим характеристикам:

- способ представления данных и знаний;
- состав используемых знаний;
- методы работы Интерпретатора.

3.3 Состав знаний экспертной системы

В общем случае знания Z некоторой группы экспертов, заносимые в БЗ, можно представить в виде двух множеств Z_1 и Z_2 ($Z= Z_1 \cup Z_2$) (рис.15.).

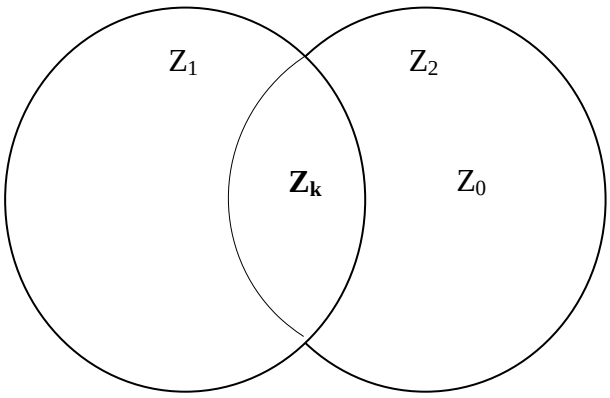


Рис.15. Природа знаний ЭС

где:
Z_1 - множество знаний, общепринятых в данной ПО и содержащихся в учебниках, справочниках, монографиях по данному вопросу (то есть, «общие» знания);
Z_2 - множество знаний, приобретенных специалистами (экспертами) в данной ПО в процессе их профессиональной деятельности («личные» знания).
Множество знаний Z_2 включает в себя подмножества личных знаний отдельных экспертов.
Пересечение Z_k множеств Z_1 и Z_2 ($Z_k= Z_1 \cap Z_2$) представляет собой канонизированную часть личных знаний - то, что усвоено экспертами из специальной литературы и в чем нет расхождений между различными экспертами.

Множество Z_0, которое не имеет пересечения с Z_1, представляет собой ту часть личных знаний, которая обусловлена профессиональным опытом и интуицией экспертов.
В ЭС традиционно особую ценность имеют слабо формализованные знания типа Z_2.
В свою очередь Z_1 - обычно хорошо структурировано и сравнительно легко формализовано.
Замечание 1. В общем случае для представления Z_1 и Z_2 в базе знаний могут применяться разные модели (способы).
Замечание 2. Любая теория - это в некотором смысле идеализация ПО, а, следовательно, и ее упрощение. Эмпирические же знания конкретны, отсюда они сложнее и многообразнее, шире и глубже описывают ПО.
Ясно, что деление знаний на множества Z_1 и Z_2 не абсолютно. Во-первых, это обусловлено наличием множества Z_k, а, во-вторых, со временем наиболее плодотворные и подтвержденные практикой гипотезы переходят из Z_0 **в** Z_k , а следовательно, в Z_1.
Состав знаний ЭС
Структуру знаний в базе знаний ЭС можно представить следующей схемой (рис.16.).

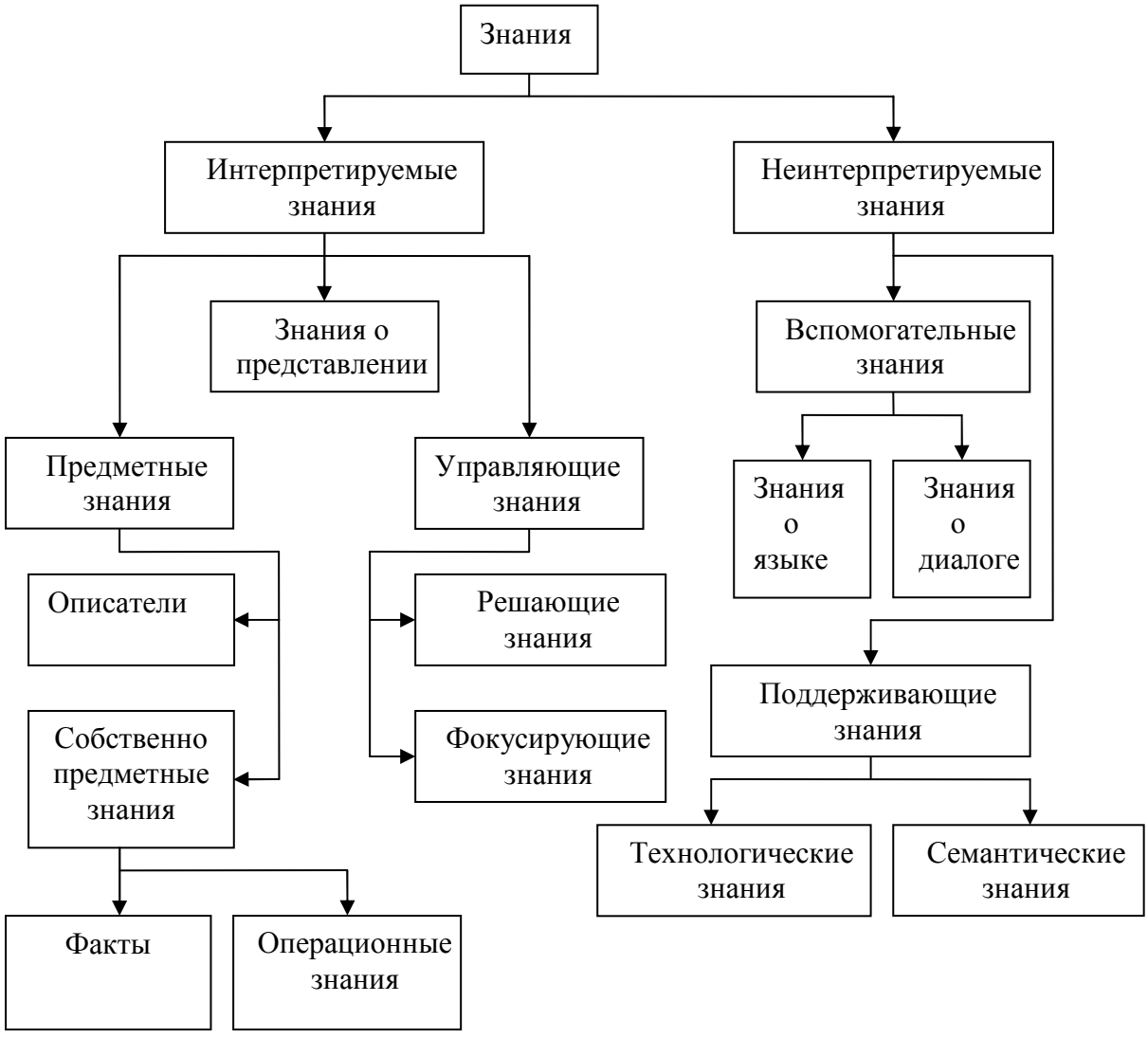


Рис.16. Структура знаний ЭС

1. К типу **неинтерпретируемых** относятся знания, которые решатель может интерпретировать («понять»). Если такие знания решателем и используются, то без «осознания» их смысла.

Вспомогательные знания. Хранят информацию о лексике и грамматике языка общения и структуре диалога; обрабатываются диалоговым процессором, но ход этой обработки решатель не осознает.

Поддерживающие знания. Используются при создании системы и при выполнении объяснений; выполняют роль описаний (обоснований) как интерпретируемых знаний, так и действий системы. **Технологические** поддерживающие знания содержат сведения о времени создания описываемых ими знаний, об авторе знаний и т.п. **Семантические** поддерживающие знания содержат смысловое описание этих знаний; они включают информацию о причинах ввода знаний, о назначении знаний, описывают способ использования знаний и получаемый эффект.

2. К **интерпретируемым** относятся знания, которые решатель способен интерпретировать.

Знания о представлении. Содержат информацию о том, каким образом (в каких структурах) в системе представлены интерпретируемые знания.

Предметные знания. Содержат данные о ПО и способах преобразования этих данных при решении поставленных задач.

- **Описатели** содержат определенную информацию о предметных знаниях (коэффициент определенности данных и правил, мера «важности», мера «сложности» и т.д.).
- **Факты** определяют возможные значения сущностей и характеристик ПО.
- **Операционные знания** содержат информацию о том, как можно изменять описание ПО в ходе решения задач; другими словами, операционные знания - это знания, задающие процедуры обработки.

Управляющие знания. Используются при функционировании решателя.

- **Фокусирующие знания** описывают, какие знания следует использовать в той или иной ситуации. Обычно содержат сведения о:
  - наиболее перспективных гипотезах; внимание фокусируется на элементах **рабочей памяти**;
  - знаниях, которые наиболее целесообразно использовать при проверке соответствующих гипотез; внимание фокусируется на элементах **базы знаний**.
- **Решающие знания** содержат информацию, используемую для выбора способа интерпретации знаний, подходящего к текущей ситуации. Эти знания используются для выбора стратегий или эвристик, наиболее эффективных для решения данной задачи.

3.3. Формальные основы экспертных систем

Напомним, что под **формальной системой F** понимается четверка:

F=(T, S, A, P), где

T - множество **базовых элементов**, исходных кирпичиков, не расчленяемых на более простые (буквы (графемы), детали в детском конструкторе и т.п.). Единственное требование к элементам множества Т состоит в том, что для любого элемента эа конечное число шагов можно узнать, принадлежит он Т или нет, а также отличить одни элементы от других, отождествляя одинаковые элементы;

S - множество **синтаксических правил**, с помощью которых из элементов **T** можно строить **синтаксически правильные конструкции** (из графем - линейно упорядоченные сочетания, называемые словами, предложениями, текстами, из деталей конструктора - макеты изделий и т.п.);

A - множество, состоящее из выделенных на основе некоторого соображения синтаксически правильных образований. Такое множество называется **начальным** или априорно принимаемым. Часто такие синтаксически правильные образования называют **аксиомами.** Тогда А называется множеством **аксиом**;

P - множество процедур, с помощью которых можно получать одни синтаксически правильные совокупности из других. Эти процедуры носят название **правил вывода,** с помощью которых можно расширять множество **A**.

Формальные системы обладают одним общим свойством - **автономностью**. Если в такой системе задать все четыре множества, она сама самостоятельно начнет генерировать множество выводимых в ней синтаксически правильных совокупностей.

Большинство ЭС базируется на понятии **формальная продукционная система**, являющимся конструктивной проекцией общего понятия формальной системы на продукционные модели.

Примечание. Продукционные системы берут свое начало с работ Поста, который в 1943 г. ввел термины **продукция** и **продукционная система.** Пост показал, что продукционная система является логической системой, эквивалентной машине Тьюринга. Другими словами, продукционные системы **универсальны**, то есть любая формальная система, оперирующая символами, может быть реализована в виде одной из продукционных систем Поста.

Система продукций Поста задается своим алфавитом $A=\{a_1, a_2, \dots, a_p\}$ и системой базисных продукций вида

$$\mathbf{x_iW \rightarrow Wy_i \ (i=1, 2, \dots, n), \quad (*)}$$

где x_i и y_i - слова в алфавите А.

Пусть некоторое слово ψ начинается словом x_i . Тогда применить к ψ продукцию (*) - это значит вычеркнуть из ψ начальный отрезок x_i и затем к оставшемуся слову приписать слово y_i.

Пример. Применив к слову **ава** продукцию авW→Wc, мы получим слово **ас**.

Каждая система продукций понимается как формальная система с правилами вывода $\mathbf{p_i}$ **(**i=1,…,n), где $\mathbf{p_i(\varphi,\psi)}$ считается **истинным** (**применимым**), если слово **ψ** получается из **φ** при помощи одной из системы продукций (*).

Наложив на набор упорядоченных продукций неявную управляющую структуру, мы переходим к понятию **нормального алгоритма** Маркова. В алгоритме Маркова упорядоченные продукции (формулы подстановок) применяются к некоторому заданному слову (аксиоме). Первая же из упорядоченных продукций, которая может быть применена к слову, применяется, изменяя слово. Затем процесс проверки применимости продукций продолжается, начиная с продукции, имеющий наивысший приоритет. Этот цикл «проверка/выполнение» продолжается до тех пор, пока либо не найдется ни одной применимой продукции, либо не будет применена некая продукция, помеченная как заключительная.

Ориентируясь на понятие продукционной системы и на структуру ЭС и ее особенности, можно формально определить продукционную систему (PS) как тройку вида:

PS=<F,P,I> , где

F - **рабочая память** системы;

P - **база знаний**, содержащая множество продукций;

I - **интерпретатор** (**решатель**), реализующий процедуры вывода.

Основные задачи, решаемые **интерпретатором**, сводятся к следующему:

- определение множества правил, которые удовлетворяются на некотором наборе текущих данных (множество **означенных правил** или **означиваний**);
- выполнение определенных означиваний с дальнейшей модификацией рабочей памяти.

В свою очередь, формальная структура **интерпретатора** может быть представлена четверкой:

I=<V, S, R, W> , где

V - процесс выбора, осуществляющий выбор из P и F подмножества активных продукций Pv и подмножества активных данных Fv, которые будут использованы в очередном цикле работы интерпретатора. Механизм выбора может быть тривиальным (на каждом цикле выбираются все правила и все данные) или более сложным для того, чтобы устранить из рассмотрения те правила, условия которых заведомо не удовлетворяются на данных рабочей памяти или малополезны.

S - процесс сопоставления, определяющий множество означиваний, то есть множество пар

<правило(p_i) - данные (d_i)>, где $p_i \in$ Pv, $d_i \in$ Fv.

Операция сопоставления может требовать много времени, поскольку в общем случае влечет за собой означивание многих переменных.

R - **процесс разрешения конфликтов** (процесс планирования), определяющий, какое из означиваний будет выполняться. Механизм разрешения конфликтов может быть **неявным** или **явным** в виде некоторого множества метаправил или процедур, описывающих выбор выполняемого правила. Метаправил позволяют обеспечить прямым и понятным способом применение динамических эвристик для разрешения конфликтов.

W - процесс, осуществляющий выполнение выбранного означенного правила (то есть, выполнение действий, указанных в правой части правила). Результатом выполнения является модификация данных в рабочей памяти F или операция ввода-вывода.

3.4. Цикл работы интерпретатора

На рис.17. представлена схема работы Интерпретатора.

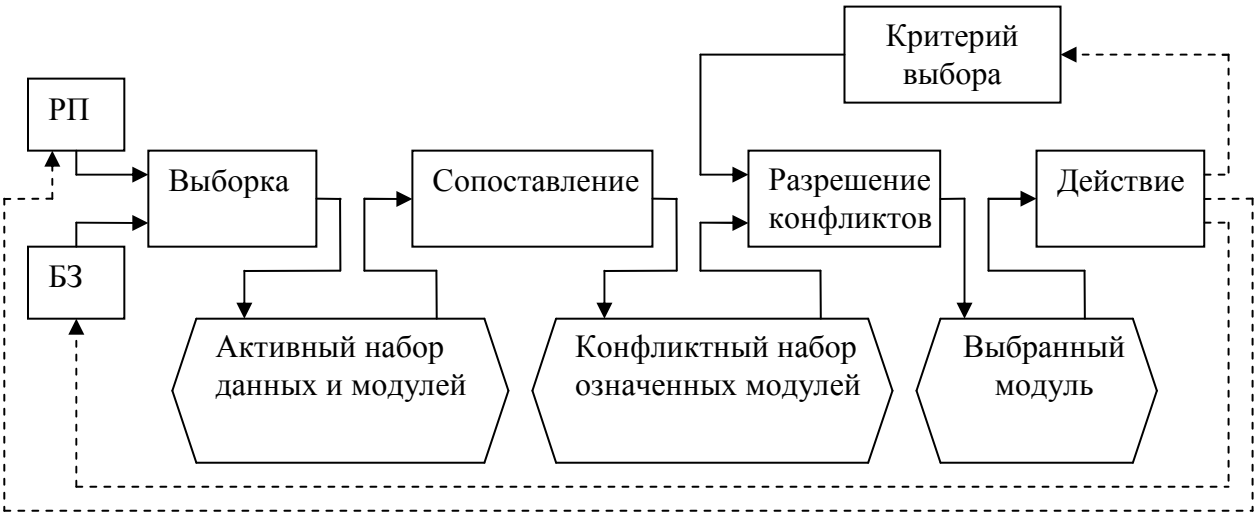


Рис.17. Общая схема работы Интерпретатора

На схеме пунктирными линиями обозначены возможные результаты выполнения действий.

Таким образом, в общем случае работа интерпретатора в каждом цикле состоит в последовательном выполнении четырех этапов:

- выборки;
- сопоставления;
- разрешения конфликтов;
- выполнения.

Этап выборки. На этом этапе осуществляется определение подмножества элементов рабочей памяти и подмножества модулей базы знаний, которые могут быть использованы в текущем цикле.

При реализации этапа выборки обычно используется один из двух подходов: **синтаксическая** выборка и **семантическая** выборка.

Синтаксическая выборка. Выполняется грубый отбор знаний (данных и/или модулей), которые могут быть полезны в текущем цикле. Основанием для выборки знаний в данном случае являются формальные (синтаксические) знания, **встроенные** в систему **разработчиком**.

Семантическая выборка. Осуществляет отбор знаний на основании таких сведений, как: модель предметной области, разбиение задачи на подзадачи, текущие цели и т.п. Семантические знания, используемые на этапе выборки, **вводятся** в систему **экспертом**, например, в виде метаправил. В результате работы этапа выборки происходит выделение активного набора даны и активного набора модулей, то есть осуществляется фокусирование внимания системы на определенном ограниченном количестве данных и модулей.

Этап сопоставления. На этом этапе определяется, какие активные модули и на каких активных данных готовы к работе. Модуль **готов к работе**, если среди активных данных есть данные, удовлетворяющие условиям этого модуля, указанным в его образце (**означенные модули**). Результатом работы этапа сопоставления является набор означенных модулей (**конфликтный** набор).

Этап разрешения конфликтов. На этом этапе интерпретатор выбирает из конфликтного набора то означивание, которое будет выполняться в текущем цикле. Интерпретатор оценивает означенные модули с точки зрения их полезности при достижении текущей цели. Подчеркивая этот факт, данный этап иногда называют **этапом планирования**.

Этап выполнения. Интерпретатор осуществляет выполнение модуля, выбранного на этапе разрешения конфликтов. В ходе этого этапа осуществляется модификация рабочей памяти, выполняются операции ввода-вывода и изменяется содержимое памяти состояний интерпретатора.

Критерий выбора. Результат выполнения модуля может включать информацию, полезную для работы интерпретатора на следующих циклах, а именно, уточнять (или изменять) критерии выбора этапа разрешения конфликтов.

Иногда цикл работы интерпретатора делят на два этапа: **распознавание** и **действие**. В этом случае в этап **распознавания** включают **выборку, сопоставление** и **разрешение конфликтов**. При этом говорят, что задача **распознавания** состоит в разрешении конфликтов.

3.5. Управление функционированием экспертной системы

С точки зрения теории работа интерпретатора зависит только от состояния рабочей памяти и состава базы знаний. На практике обычно учитывается **история** работы интерпретатора, то есть поведение интерпретатора в предшествующих циклах.

Информация о поведении интерпретатора запоминается в его **памяти состояний**. Таким образом, в общем виде схему управления функционированием ЭС можно представить в следующем виде (рис.18.).

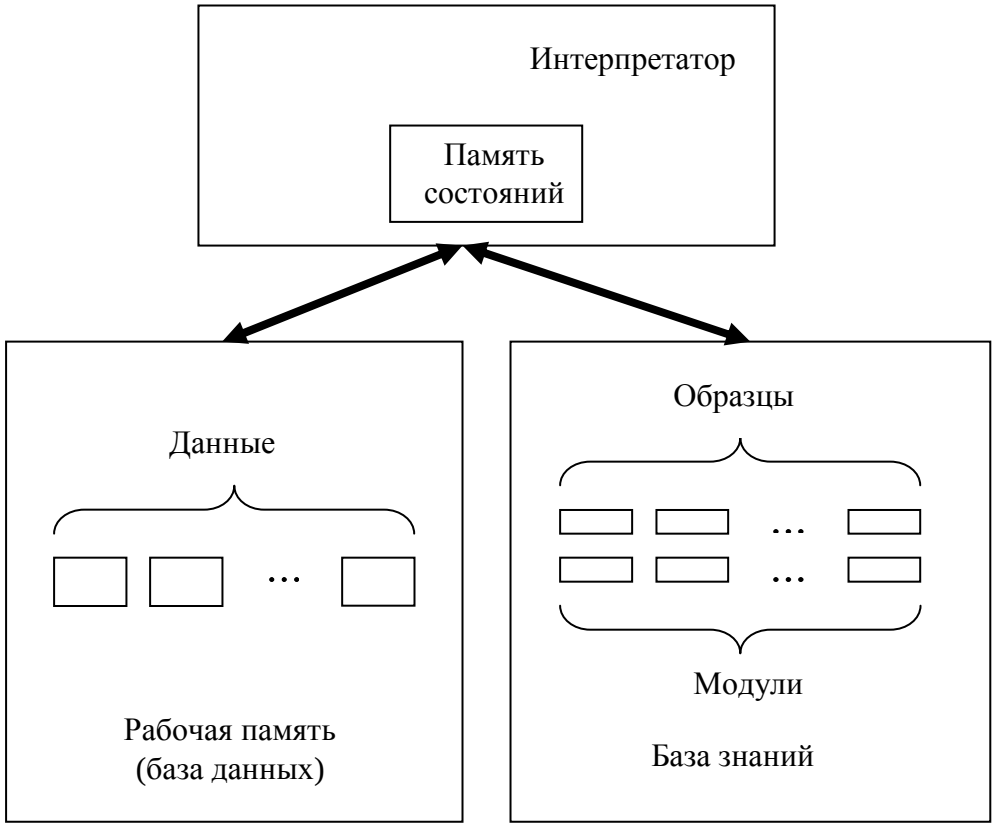


Рис.18. Общая схема управления функционированием ЭС

Обычно **память состояний** содержит в том или ином виде **протокол** работы системы. Таким образом, в общем случае каждый из этапов работы интерпретатора использует три источника знаний: **рабочую память, базу знаний** и **память состояний**.

Основное отличие управляющей компоненты ЭС от традиционных механизмов управления выводом состоит в следующем:

- отдельные модули вызывают не по имени, а по описанию ситуации;
- способ взаимосвязи модулей формируется в процессе решения задачи, так как выбор очередного модуля зависит от текущей ситуации и не может быть сформирован заранее.

В действительности ЭС не располагают процедурами, которые могли бы построить в пространстве состояний сразу весь путь решения задачи. Более того, зачастую даже не удается определить, имеется ли вообще какое-либо решение задачи. Тем не менее поиск решения выполняется, поскольку движением в пространстве состояний управляют **скрытые** (**виртуальные**) процедуры - **демоны** (или **недерминированные** процедуры**)**. Демоны активизируются только тогда, когда их просят о помощи.

Другими словами, компонент вывода обладает способностью функционировать в условиях недостатка информации, то есть способен продолжать рассуждения и со временем найти решение, может быть, и не точное; система никогда не должна останавливаться из-за отсутствия какой-либо части входной информации.

Замечание. Во фреймовых системах управление осуществляется **демонами**, входящими в состав **слотов**.

Полученные в результате срабатывания продукций новые знания могут использоваться в следующих целях:

- понимание и интерпретация фактов и правил с использованием продукций, фреймов, семантических сетей;
- решение задач с помощью моделирования;
- идентификация источника данных, причин несовпадений новых знаний со старыми, получение метазнаний;
- составление вопросов к системе;
- усвоение новых знаний, устранение противоречий, систематизация избыточных данных.

Процесс рассмотрения системой набора правил (выполнение программы) называют **консультацией**. Ее наиболее удобная для пользователя форма - дружественный диалог с компьютером.

Диалог может быть построен на системе вопросов, задаваемых пользователем, экспертной системой, или фактов-данных, хранящихся в БД.

При прямом поиске пользователь может задавать две группы вопросов, на которые компьютер дает объяснения:

- КАК получено решение; при этом компьютер должен выдать на экран трассу в виде ссылок на использованные правила;
- ПОЧЕМУ система задала какой-то вопрос; при этом на экран выдается своеобразная трасса, которую система хотела бы использовать для вывода после получения ответа на задаваемый пользователем вопрос.

3.6. Ведение диалога в диагностический ЭС

При построении ЭС возникает проблема организации диалога с пользователем. Диалог должен быть организован таким образом, чтобы задаваемые вопросы поступали пользователю в нужное время и выглядели бы естественными для сложившейся в процессе вывода ситуации. Поэтому техника ведения диалога должна быть тщательно продумана на стадии создания ЭС. Один из известных подходов к созданию диагностической процедурной ЭС выглядит следующим образом.

Задано множество фактов $\mathbf{A = \{a_{ij}\} \cup \{q_i\} = (a_1, a_2, \ldots, a_n)}$, состоящее из элементов двух типов. Элементы a_{ij} определяют обычные декларативные знания из конкретной ПО. Элементы q_i определяют два вида взаимодействия с внешней средой:

- представляют собой вопросы пользователю в виде альтернативного меню:

$$q_i = a_{i1}, a_{i2}, \ldots, a_{in}.$$

- являются результирующим заключением (или диагнозом), оформленным в виде соответствующих сообщений пользователю.

Продукции в данной системе имеют вид

$$a_{ij} \rightarrow q_m = \{a_{m1}, a_{m2}, \ldots, a_{mk}\}.$$

Все множество фактов и продукций организовано в некоторую систему, представленную в виде графа “ИЛИ”. Фрагмент такого диагностирующего графа с вершинами-диагнозами q_9, q_{10}, q_{11}, q_{12} (терминальными вершинами) представлен на следующем рисунке.

Принцип работы такой ЭС заключается в следующем.

После обращения пользователя к ЭС происходит попадание в вершину q_1, инициирующую вопрос пользователю в виде соответствующего альтернативного меню:

$$q_1 = \{a_{12}, a_{13}, \ldots\}.$$

Допустим, на вопрос системы: “Какой из фактов a_{12}, a_{13}, … имеет место?” пользователь ответил: a_{12}. В результате факт a_{12} заносится в рабочее поле и вывод продолжается с вершины – вопрос q_2 . В конечном счете процесс вывода достигает одной из терминальных вершин, соответствующей предполагаемому диагнозу.

При такой структуре в ЭС достаточно просто может быть реализована ***подсистема объяснения*** как важнейшая часть любой ЭС. Для этого достаточно каждой вершине q_i графа сопоставить соответствующий текст, описывающий мотивации выбора в данной вершине. А далее происходит вывод этих текстов при движении снизу (от результата) вверх в соответствии с реализованной цепочкой рассуждений.

Для ответа на вопросы пользователя типа: ”Почему q_i, а не q_j?” система, двигаясь по цепочке вывода снизу вверх, определяет место разветвления (какую-то вершину q_k) путей, ведущих к q_i и q_j. Пояснительный текст, ассоциированный с вершиной q_k, и должен содержать ответ пользователю. Например, на вопрос “Почему q_9, а не q_{12}?” система ответит “Потому что a_{12}, а не a_{13}.” . Такая структуризация базы знаний ЭС представлена на рис.19.

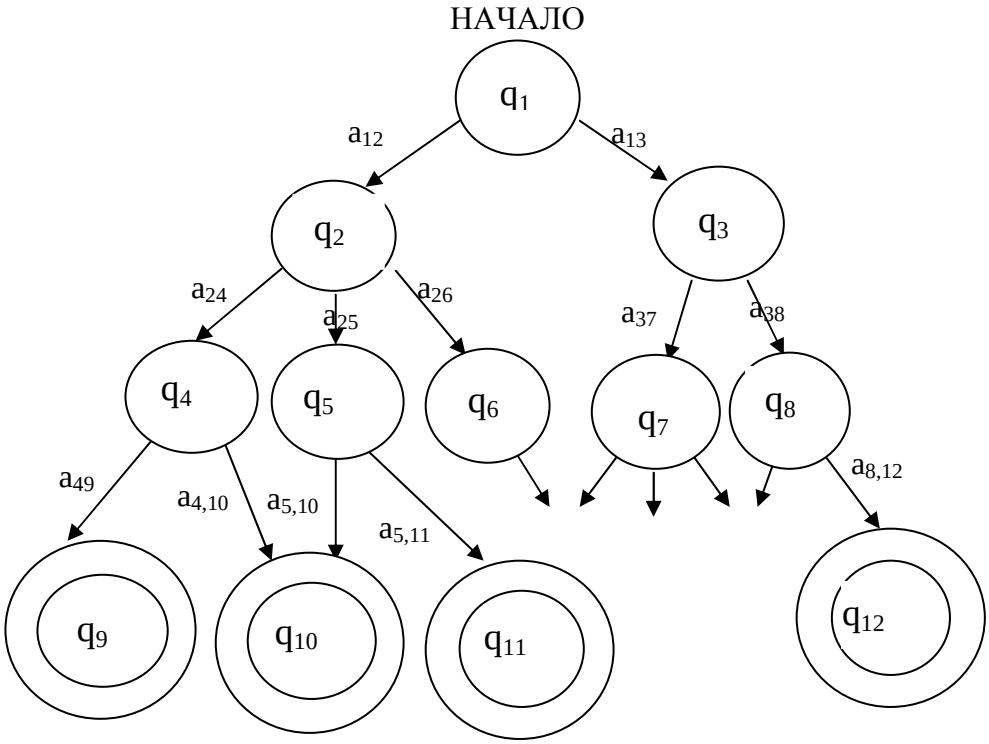


Рис. 19. Структура базы знаний диагностической ЭС

Такая структуризация в данном случае оказывается более естественной и логически оправданной, чем, например, непосредственное использование продукций, построенных в

соответствии с различными путями, ведущими от начала процесса к каждой из терминальных вершин, например:

$$a_{12} \wedge a_{24} \wedge a_{49} \rightarrow q_9;$$
$$a_{12} \wedge a_{24} \wedge a_{4,10} \rightarrow q_{10};$$
$$a_{12} \wedge a_{25} \wedge a_{5,10} \rightarrow q_{10}.$$

3.7. Характеристики экспертных систем

Выделим следующие характеристики ЭС:

- назначение,
- проблемная область,
- глубина анализа проблемной области,
- тип используемых методов и знаний,
- класс системы,
- стадия существования,
- инструментальные средства.

1. **Назначение** определяется следующей совокупностью параметров:
   - цель создания экспертной системы — для обучения специалистов, для решения задач, для автоматизации рутинных работ, для тиражирования знаний экспертов и т.п.;
   - основной пользователь — не специалист в области экспертизы, специалист, учащийся.
2. **Проблемная область** может быть определена совокупностью **параметров предметной области** и **задач**, решаемых в ней.

   Каждый из **параметров** можно рассматривать как с точки зрения **конечного пользователя**, так и **разработчика** экспертной системы.

   А) С точки зрения пользователя:

   предметную область можно характеризовать ее описанием в терминах пользователя, включающим:
   - **наименование** области,
   - **перечень и взаимоотношения подобластей** и т.п.;

   задачи, решаемые существующими экспертными системами, — их **типом**. Обычно выделяют следующие типы задач:
   - интерпретация символов или сигналов — составление смыслового описания по входным данным;
   - диагностика — определение неисправностей (заболеваний) по симптомам;
   - предсказание — определение последствий наблюдаемых ситуаций;
   - конструирование — разработка объекта с заданными свойствами при соблюдении установленных ограничений;
   - планирование — определение последовательности действий, приводящих к желаемому состоянию объекта;
   - слежение — наблюдение за изменяющимся состоянием объекта и сравнение его показателей с установленными или желаемыми;
   - **управление — воздействие на объект для достижения желаемого поведения.**

   В) С точки зрения разработчика целесообразно выделять **статические** и **динамические** предметные области.

   Предметная область называется **статической**, если описывающие ее исходные данные **не изменяются во времени** (точнее, рассматриваются как не изменяющиеся за время решения задачи). Статичность области означает неизменность описывающих ее исходных данных. При этом производные данные (выводимые из исходных) могут и появляться заново, и изменяться (не изменяя, однако, исходных данных).

Если исходные данные, описывающие предметную область, **изменяются** за время решения задачи, то предметную область называют **динамической**.

Кроме того, предметные области, с точки зрения разработчика, можно характеризовать следующими аспектами:

- числом и сложностью сущностей, их атрибутов и значений атрибутов,
- связностью сущностей и их атрибутов;
- полнотой знаний; точностью знаний (знания точны или правдоподобны: правдоподобность знаний представляется некоторым числом или высказыванием).

Решаемые задачи, с точки зрения разработчика экспертной системы, также можно разделить на **статические** и **динамические**.

Будем говорить, что ЭС решает динамическую или статическую задачу, если процесс ее решения изменяет или не изменяет исходные данные о текущем состоянии предметной области.

В подавляющем большинстве существующие ЭС исходят из предположения о **статичности предметной области** и решают **статические задачи**. Будем называть такие ЭС **статическими**.

ЭС, которые имеют дело **с динамическими предметными областями** и решают **статические** или **динамические задачи**, будем называть **динамическими**.

Решаемые задачи, кроме того, могут характеризоваться следующими аспектами:

- **числом и сложностью правил**, используемых в задаче,
- **связностью** правил,
- **пространством поиска**,
- **классом** решаемых **задач**.

По **степени сложности** выделяют **простые** и **сложные** правила. К **сложным** относятся правила, текст записи которых на естествен ном языке занимает 1/3 страницы и больше. Правила, текст записи которых занимает менее 1/3 страницы, относят к **простым**.

Можно сказать, что **степень сложности задачи** определяется не просто общим числом правил данной задачи, а числом правил в ее наиболее связной независимой подзадаче.

Пространство поиска может быть определено по крайней мере тремя факторами: **размером**, **глубиной** и **шириной**.

- **Размер** пространства поиска дает обобщенную характеристику сложности задачи. Выделяют **малые** (до $3{,}6 * 10^6$ состояний) и **большие** (свыше $3{,}6 * 10^6$ состояний) пространства поиска.
- **Глубина** пространства поиска характеризуется средним числом последовательно применяемых правил, преобразующих исходные данные в конечный результат.
- **Ширина** пространства — средним числом правил, при годных к выполнению в текущем состоянии.

Класс задач определяет методы, используемые ЭС для их решения. В рамках ЭС различают задачи **расширения**, **доопределения**, **преобразования**.

- К **задачам расширения** относятся такие, в процессе решения которых осуществляется **только увеличение** информации о предметной области, не приводящее ни к изменению ранее выведенных данных, ни к другой области задач. Это **статические задачи на статических** ПО.
- К **задачам доопределения** относятся задачи **с неполной** или **неточной** информацией о реальной предметной области, цель решения которых — выбор из множества альтернативных текущих со стояний предметной области того, которое адекватно исходным данным. В случае **неточных** данных альтернативные текущие состояния возникают как **результат ненадежности** данных и правил, что приводит к многообразию различных доступных выводов из одних и тех же исходных данных. В случае **неполных** данных альтернативные состояния являются результатом **доопределения** области, то есть результатом предположений о **возможных**

значениях недостающих данных**.** Задачи доопределения - **динамические** на **статических** ПО.

- К задачам **преобразования** относятся задачи, осуществляющие изменение исходной или выведенной ранее информации о ПО. Эти изменения являются следствием изменений либо реального мира, либо его модели. Задачи **преобразования** являются **динамическими** на динамических ПО.

3. **Глубина анализа** проблемной области. По степени сложности структуры ЭС делят на **поверхностные** и **глубинные**.
   Поверхностные ЭС представляют знания об области экспертизы в виде правил (условие—действие). Условие каждого правила определяет образец некоторой ситуации, при соблюдении которой правило может быть выполнено. Поиск решения состоит в выполнении тех правил, образцы которых сопоставляются с текущими данными (текущей ситуацией в РП). При этом предполагается, что в процессе поиска решения последовательность формируемых таким образом ситуаций не оборвется до получения решения, т.е. не возникнет неизвестной ситуации, которая не сопоставится ни с одним правилом.
   Глубинные ЭС, кроме возможностей поверхностных систем, обладают способностью при возникновении неизвестной ситуации определять с помощью некоторых общих принципов, справедливых для области экспертизы, какие действия следует выполнить.
4. **По типу используемых методов и знаний** ЭС делят на **традиционные** и **гибридные**.
   Традиционные ЭС используют в основном неформализованные методы инженерии знаний и неформализованные знания, полученные от экспертов.
   Гибридные ЭС используют методы инженерии знаний, формализованные методы, а также данные традиционного программирования и математики.
5. **Классы систем**. Совокупность рассматриваемых выше характеристик позволяет определить особенности конкретной ЭС. Однако пользователи зачастую стремятся охарактеризовать ЭС каким-либо одним **обобщенным параметром.** В этой связи говорят о **поколениях** ЭС:
   - **первое поколение** — статические поверхностные ЭС;
   - **второе поколение** — статические глубинные ЭС (иногда ко второму поколению относят гибридные ЭС);
   - **третье поколение** — динамические ЭС (глубинными и гибридными).

   Иногда выделяют два больших класса ЭС (существенно отличающихся по технологии их проектирования), условно называемые **простыми** и **сложными** ЭС.
   К **простым** можно отнести **поверхностную** и **традиционную** (реже гибридную) ЭС, выполненные на персональной ЭВМ и содержащие от 200 до 1000 правил.
   К **сложным** ЭС относятся **глубинная** и **гибридная** ЭС, вы полненные либо на символьной, либо на мощной универсальной ЭВМ, либо на интеллектуальной рабочей станции, содержащие от 1500 до 10000 правил.
6. **Стадия существования** характеризует степень проработанности и отлаженности ЭС. Обычно выделяют следующие стадии: **демонстрационный прототип**, **исследовательский прототип**, **действующий прототип**, **промышленная система**, **коммерческая система**.
   Демонстрационным прототипом называют ЭС, которая решает часть требуемых задач, демонстрируя жизнеспособность метода инженерии знаний. При наличии развитых интеллектуальных систем для разработки демонстрационного прототипа требуется примерно 1—2 мес. демонстрационный прототип работает, имея 50—100 правил. Развитие демонстрационного прототипа приводит к исследовательскому прототипу.
   Исследовательским прототипом называют систему, которая решает все требуемые задачи, но неустойчива в работе и не полностью проверена. Исследовательский прототип обычно имеет в базе знаний 200 — 500 правил, описывающих проблемную область.
   Действующий прототип надежно решает все задачи, но для решения сложных задач может потребоваться чрезмерно много времени и (или) огромная память. Число правил в такой системе равно 500—1000.

Экспертная система, **достигшая промышленной стадии**, обеспечивает высокое качество решения всех задач при минимуме времени и памяти. Обычно процесс преобразования действующего прототипа в промышленную систему состоит в расширении числа правил до 1000— 1500 и переписывания программ с использованием более эффективных интеллектуальных систем.

Обобщение задач, решаемых на стадии промышленной системы, позволяет перейти к стадии **коммерческой системы**, пригодной не только для собственного использования, но и для продажи раз личным потребителям. В базе знаний такой системы 1500—3000 правил.

7. Диапазон возможных **средств построения** ЭС простирается от языков высокого уровня до средств поддержки низкого уровня. Разделим инструментальные средства построения ЭС на четыре основных категории:
   - языки программирования;
   - языки инженерии знаний;
   - вспомогательные средства;
   - средства поддержки.

Языки программирования, применяемые для работы в области ЭС, — это, как правило, или **проблемно-ориентированные** языки (Фортран, Паскаль и т.д.), или языки **обработки текстов** (Лисп, Пролог). Проблемно-ориентированные языки разработаны для специального класса задач. Например, Фортран удобен для выполнения алгебраических вычислений и чаще всего применяется в научных, математических и статистических вычислениях. Языки обработки текстов разработаны для прикладных областей искусственного интеллекта. Например, Лисп имеет механизмы для манипулирования символами в форме списковых структур. Список является просто набором элементов, заключенных в скобки, где каждый элемент может быть или символом, или другим списком. Списковые структуры являются удобным строительным материалом для представления сложных понятий. В языке Лисп все отношения между объектами описываются через списки, содержащие отношения объекта с другими объектами.

Языки программирования, подобные Лиспу, представляют максимальную гибкость разработчику ЭС, но никак не подсказывают ему, как представлять знания или как построить механизм доступа к базе знаний.

Языки инженерии знаний обладают меньшей гибкостью, чем языки программирования, поскольку разработчик системы должен пользоваться схемой управления, определяемой встроенным в язык механизмом вывода. Эти языки, однако, обеспечивают некоторое руководство и готовые механизмы вывода для управления и использования базы знаний.

Язык инженерии знаний является искусным инструментальным средством разработки ЭС, погруженным в обширное поддерживающее окружение. Языки инженерии знаний можно разделить на **скелетные** и **универсальные**.

Скелетный язык инженерии знаний является просто «раздетой» экспертной системой без специальных предметных знаний, включающей в себя только механизм вывода и средства поддержки.

Универсальный язык инженерии знаний может быть применим к проблемам разного типа в различных прикладных областях. Он обеспечивает более широкие возможности управления поиском данных и доступом к ним, чем скелетные системы, но может оказаться, что его труднее использовать. Разные универсальные языки значительно варьируют в смысле общности и гибкости.

Вспомогательные средства построения ЭС состоят из программ, оказывающих помощь в приобретении знаний у эксперта, и представлении их, и программ, которые помогают разрабатывать проекты экспертных систем.

Средства поддержки — это просто пакеты программ, которые прилагаются к средству построения ЭС, чтобы упростить его использование, облегчить диалог и сделать его более эффективным. Это — средства отладки, ввода-вывода, объяснения, редакторы баз знаний.

4. НЕЧЕТКИЕ ЗНАНИЯ

4.1. Основные понятия

При попытке формализовать человеческие знания исследователи вскоре столкнулись с проблемой, затруднявшей использование традиционного математического аппарата для их представления. Существует целый класс описаний, оперирующих качественными характеристиками объектов (**много**, **мало**, **сильный**, **очень сильный** и т.п.). Эти характеристики обычно **размыты** и не могут быть однозначно интерпретированы, однако содержат важную информацию (**высокая** температура - один из возможных признаков гриппа). Кроме того, в задачах, решаемых интеллектуальными системами, часто приходится пользоваться неточными знаниями, которые не могут быть интерпретированы как **полностью истинные** или **ложные**, то есть, существуют знания, достоверность которых выражается некоторой промежуточной цифрой, например, 0,7. Возникает очевидная проблема - как, не разрушая свойства размытости и неточности, представлять подобные знания формально?
В начале 1970-х годов Лотфти Заде (США) предложил формальный аппарат **нечеткой алгебры** и **нечеткой логики**. Позднее это направление положило начало одной из ветвей искусственного интеллекта - **мягким вычислениям**.

4.1.1. Основы теории нечетких множеств

Согласно Заде, назовем **лингвистической** переменную (ЛП), значения которой определяются набором вербальных (словесных) характеристик некоторого свойства.
Например, ЛП «рост» определяется через набор значений:
{**карликовый**, **низкий**, **средний**, **высокий**, **очень высокий**}
В свою очередь, сами значения ЛП определяются через, так называемые, **нечеткие множества** (НМ), которые, в свою очередь, определяются на некоторой **базовой числовой шкале**, имеющей размерность соответствующей ЛП.
Пусть U - базовая шкала (*универсальное множество*, *универсум*), определенная для некоторого НМ X (точнее, для ЛП, значением которой является X).
Говорят, что X **задано**, если задана **функция принадлежности $\mu_B(u)$**, $\forall u \in U$, принимающая значения на [0,1].
То есть, НМ X - это совокупность пар вида **$(\mu_x(u),u)$**, где $u \in U$.
Таким образом, для задания нечеткого множества необходимо задать **базовую шкалу** и **функцию принадлежности**.
Функция принадлежности определяет субъективную **степень уверенности** эксперта в том, что данное конкретное значение базовой шкалы соответствует определяемому НМ.
Замечание. Функцию принадлежности не следует путать с вероятностью, носящей **объективный** характер.
Множество элементов $u \in U$, для которых **$\mu_x(u)>0$,** называют **носителем** НМ X.
Составляющие множества **носитель=$\{u_1, u_2, \dots u_n\}$** (элементы **$u_1, u_2, \dots u_n$**) могут быть упорядоченными числами, интервалами, понятиями.
Часто НМ с дискретным носителем представляется суммой вида:

$$\mathbf{X=\mu_1/u_1+\mu_1/u_1+\dots+\mu_1/u_1=}\sum_j \mu_j/u_j$$

Здесь символ «/» отделяет функцию принадлежности от носителя, а знак «+» обозначает объединение, то есть с использованием операции объединения

$$\mathbf{X=}\bigcup \mu_j/u_j$$

Пример. Определим «возраст» как ЛП со множеством значений
{**младенческий**, **детский**, **юный**, **молодой**, **зрелый**, **преклонный**, **старческий**}
Пусть для ЛП «возраст» базовая шкала представлена числовой шкалой [0,120].

Определим функцию принадлежности как степень уверенности в том, что данное количество лет можно отнести к конкретной возрастной категории (значению ЛП). Мнение эксперта по определению НМ, соответствующих значениям **младенческий** и **детский**,
может выглядеть следующим образом (рис.20.).

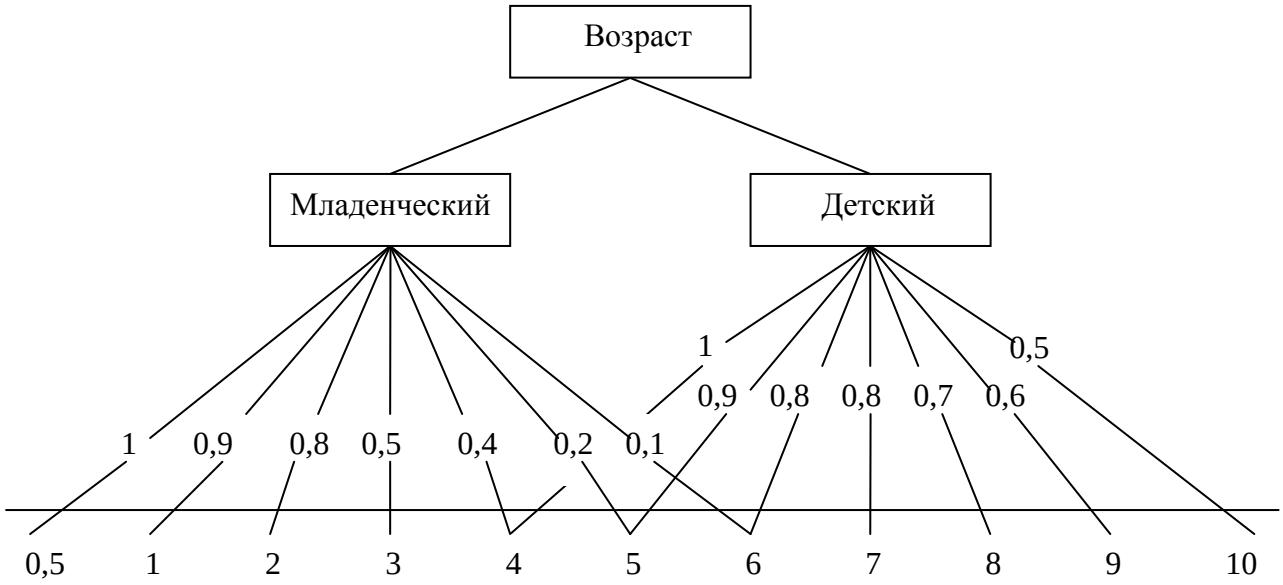


Рис.20. Пример определения нечетких множеств

НМ «младенческий» возраст можно определить в следующих форматах:
Младенческий = {(1,0.5), (0.9,1), (0.8,2), (0.5,3), (0.4,4), (0.2,5), (0.1,6)}
Младенческий = {1/0.5, 0.9/1, 0.8/2, 0.5/3, 0.4/4, 0.2/5, 0.1/6},
Младенческий = 1/0.5+0.9/1+0.8/2+0.5/3+0.4/4+0.2/5+0.1/6.

4.1.2. Операции с нечеткими множествами

Способы, с помощью которых определяются операции с нечеткими знаниями, выраженными при помощи ЛП, являются, в основном, эвристическими.
Равенство. Два НМ X и Y **равны** (X=Y) тогда и только тогда, когда $\mu_X(u)= \mu_Y(u)$ $\forall u \in U$.
Замечание. Далее, если не оговорено отдельно, для краткости формулу
$\mu_X(u) \Theta \mu_Y(u)$ $\forall u \in U$, где Θ - некоторая операция**,** будем заменять формулой $\mu_X \Theta \mu_Y$
Включение. НМ X **содержится** в НМ Y или **является подмножеством** Y ($X \subset Y$) тогда и только тогда, когда $\mu_X \leq \mu_Y$.
Дополнение. Говорят, что X' есть дополнение к X тогда и только тогда, когда $\mu_{X'}=1 - \mu_X$.
Пересечение $X \cap Y$
Пересечение двух НМ X и Y определяется как наибольшее НМ множество, содержащееся как в X, так и в Y. Функция принадлежности для $X \cap Y$ определяется как $\mu_{X \cap Y} = \mathbf{Min}(\mu_X, \mu_Y)$.
Здесь союз «и» понимается в **жестком смысле**, так как либо $\mu_X > \mu_Y$, либо $\mu_X < \mu_Y$.
То есть отсутствует возможность каких-либо «компенсаций» имеющихся значений $\mu_X(x)$ какими-либо значениями $\mu_Y(x)$ и наоборот**.** Например, если $\mu_X(x)=0,8$, а $\mu_Y(x)=0,5$, то $\mu_{X \cap Y}(x)=0,5$.
В некоторых случаях ближе к смыслу союза «и» может оказаться его **мягкая** интерпретация, соответствующая образованию алгебраического произведения $\mu_X \cdot \mu_Y$.
Объединение $X \cup Y$
Объединение двух НМ X и Y определяется как наименьшее НМ множество, содержащее как X, так и Y. Функция принадлежности для $X \cup Y$ определяется как $\mu_{X \cup Y} = \mathbf{Max}(\mu_X, \mu_Y)$.
Или в более простом **жестком** виде $\mu_{X \cup Y} = \mu_X \vee \mu_Y$.

В **мягком** смысле операции объединения соответствует операция **X⊕Y** - алгебраическая сумма:
$\mu_{X\oplus Y}(x)= \mu_X+\mu_Y - \mu_X \cdot \mu_Y$.
Легко доказать следующее соотношение: $X\cup Y=(X'\cap Y')'$
Доказательство:
$X\cup Y = Max (\mu_X, \mu_Y)$
$(X'\cap Y')' = 1 - Min (\mu_{X'}, \mu_{Y'}) = 1 - Min (1-\mu_X, 1-\mu_Y)$
1) $\mu_X > \mu_Y$
$Max (\mu_X, \mu_Y)= \mu_X$
$1 - Min (1-\mu_X, 1-\mu_Y)=1 - (1-\mu_X) = \mu_X$.
Аналогично при $\mu_X < \mu_Y$.
Пример. Трактовка в **жестком** смысле операций пересечения и объединения, когда X («младенческий») и Y («детский») - НМ, принадлежащие одной ЛП.

Годы	0,5	1	2	3	4	5	6	7	8	9	10
$\mu_{млад}$	1	0,9	0,8	0,5	0,4	0,2	0,1	0	0	0	0
$\mu_{дет}$	0	0	0	0	1	0,9	0,8	0,8	0,7	0,6	0,5
$\mu_{X\cap Y}$	0	0	0	0	0,4	0,2	0,1	0	0	0	0
$\mu_{X\cup Y}$	1	0,9	0,8	0,5	1	0,9	0,8	0,8	0,7	0,6	0,5

Алгебраические операции

1. Произведение нечетких множеств

В теории нечетких множеств существует несколько схем выполнения операции умножения нечетких множеств.
1). Алгебраическое произведение НМ X и Y (**X∗Y**) определяется следующим образом:

$$X*Y=XY=\{\mu_X \cdot \mu_Y\}$$

Частными случаями операции произведения являются операции:
A) умножения нечеткого множества на константу **k** ($0\le k\le 1$) - $k\cdot X=\{ k\cdot\mu_X \}$
B) возведения множества в степень (операция концентрирования - сужения диапазона определения нечеткого множества) - $CON(X)=\{(\mu_X)^2\}$
C) растяжения(расширения диапазона определения нечеткого множества):

$$DIL(X)=\{(\mu_X)^{0.5}\}$$

Примеры. Пусть
U={1,2,3,4,5}; X={0.5/2, 1/3, 0.1/5}; Y={0.3/2, 1/4, 0.3/5}. Тогда:
X∗Y={0.15/2, 0.03/5};
0,9·X={0.45/2, 0.9/3, 0.09/5};
CON(X)={0.09/2, 1/4, 0,09/5};
DIL(X)={0.55/2, 1/4, 0.55/5}.
2). Граничное произведение X⊗Y определяется как $\mu_{X\otimes Y}=max((\mu_X + \mu_Y -1),0)$
3). Драстическое (радикальное, решительное) **произведение X ∧ Y:**

$$\mu_{X\wedge Y}= \begin{cases} \mu_Y, \text{ если } \mu_X=1, \\ \mu_X, \text{ если } \mu_Y=1, \\ 0, \text{ в противном случае.} \end{cases}$$

2. Суммирование нечетких множеств

1). Алгебраическая сумма X+Y определяется следующим образом:

$$\mu_{X+Y} = \mu_X + \mu_Y - \mu_X \cdot \mu_Y$$

2). Граничная сумма X⊕Y определяется как $\mu_{X\otimes Y} = \max((\mu_X + \mu_Y),1)$

3). Драстическая сумма X∇Y:

$$\mu_{X\nabla Y} = \begin{cases} \mu_Y, \text{ если } \mu_X=0, \\ \mu_X, \text{ если } \mu_Y=0, \\ 1, \text{ в противном случае.} \end{cases}$$

4) Взвешенная сумма (λ-сумма) X_λ+Y, где **λ** - весовой коэффициент.

$$\mu_{X\lambda+Y} = \lambda\mu_X + (1-\lambda)\mu_Y$$

2. Разность нечетких множеств

В ряде практических приложений определяют функции принадлежности разности и модуля разности пары нечетких множеств.

Разность нечетких множеств X-Y = X∩Y':

$$\mu_{X-Y} = \max((\mu_X + \mu_Y, 0)) \text{ или } \mu_{X-Y} = \min(\mu_X, 1- \mu_Y);$$

3. Оператор увеличения нечеткости

Предназначен для преобразования четкого множества в нечеткое либо увеличения нечеткости нечеткого множества. Результатом его действия служит объединение семейства множеств μ_X***K:**

$$F(X,K) = \cup\mu_X \cdot K, \text{ где}$$

μ_X - число или значение функции исходного нечетного множества ;

K - ядро оператора увеличения нечеткости, выраженное нечетким множеством.

Пример. Пусть **U={0, 1, 2, 3}; X={0.9/0, 0.6/1, 0.3/2, 0/3}=0.9/0+0.6/1+0.3/2+0/3**

K(0)={1/0, 0.5/1}=1/0+ 0.5/1;

K(1)={1/1, 0.6/2, 0.3/3}=1/1+0.6/2+ 0.3/3;

K(2)={1/2, 0.5/3}=1/2+ 0.5/3;

K(3)={1/3}.

Тогда:

F(X,K)= μ_X(0)K(0) ∪μ_X(1)K(1) ∪μ_X(2)K(2) ∪μ_X(3)K(3) =

=0.9(1/0+0.5/1) ∪0.6(1/1+0.6/2+0.3/3) ∪0.3(1/2+0.5/3) ∪0.0(1/3)=

=(0.9/0+0.45/1) ∪(0.6/1+0.36/2+0.18/3) ∪(0.3/2+0.15/3)=

0.9/0+max(0.45/1,0.6/1)+max(0.35/2,0.3/2)+max(0.18/3,0.15/3)=0.9/0+0.6/1+0.36/2+0.18/3

Сравнение исходного множества X и результата действия оператора F показывает,

что синглетоны с числами 2 и 3 имеют увеличенные значения функции принадлежности для нечеткости множества X.

4.2. Алгебра нечетких отношений

4.2.1. Основные понятия

Понятие нечеткого отношения является одним из основных в теории нечетких множеств, т.к. на его основе формулируются и анализируются математические модели многих реальных сущностей, выполняются нечеткие преобразования и нечеткие логические выводы. С их помощью можно наиболее полно отражать степень различных отношений, возникающих у реальных объектов друг к другу.

Под нечетким n-арным отношением R (n-отношением) на декартовом произведении n универсальных множеств (базовых шкал) $U=U_1\times U_2\times\ldots\times U_n = \{(u_1,u_2,\ldots,u_n)\ u_i\in U_i,\ i=1\ldots n\}$ понимается нечеткое множество

$$R=X_1\times X_2\times\ldots\times X_n = \{\mu_R(u_1,u_2,\ldots,u_n)/(u_1,u_2,\ldots,u_n)\}, \text{ где}$$

$\mu_R(u_1,u_2,\ldots,u_n)$ - функция принадлежности отношения. Она выражает силу зависимости между элементами $u_1,u_2,\ldots,u_n$, $u_i \in U_i$, $i=1\ldots n$. $\mu_R: U \to [0,1]$.

В частном случае функцию принадлежности μ_R можно интерпретировать как

$$\mu_R(u_1,u_2,\ldots,u_n) = \min(\mu_{X1}(u_1), \mu_{X2}(u_2), \ldots, \mu_{Xn}(u_n)).$$

Нечеткое бинарное отношение R *фазимножеств* A и B (A⊂U и B⊂V , где U=$\{u_1,u_2,\ldots,u_n\}$, V=$\{v_1,v_2,\ldots,v_n\}$ - носители нечетких множеств, например, выражаемых фразами "A значительно больше B", либо "A примерно равно B", которые называют обычно отношениями предпочтения) на прямом произведении пространств UxV = {(u,v)| u∈U, v∈V}есть нечеткое множество вида

$$R: A \to B;\ R:(u,v) \to [0,1] \text{ или}$$

$$R=\{\mu_{11}/(u_1,v_1), \mu_{12}/(u_1,v_2), \ldots, \mu_{1n}/(u_1,v_n), \ldots, \mu_{mn}/(u_m,v_n)= \sum_{j=1}^{m}\sum_{k=1}^{n} \mu_R(u_j,v_k)/ (u_j,v_k)$$

с функцией принадлежности $\mu_R: U \times V \to [0,1]$

Значение $\mu_R(u_j,v_k)$ для пары $(u_j,v_k) \in U \times V$ характеризует степень выполнения (либо уверенности, предпочтения) отношения $u_j R v_k$. Очевидно, что обычное отношение есть частный случай нечеткого отношения с функцией принадлежности, принимающей значение 0 или 1.

Нечеткие отношения могут быть представлены различными понятиями, описанными нечеткими множествами: "близко_к", "имеет_сходство", "много_больше" и т.п.

Пример 1. Пусть X - НМ «высокий» с функцией принадлежности

$\mu_X = \{0.7/180, 0.8/185, 0.9/190, 1.0/195, 1.0/200\}$

и Y - НМ «тяжелый» с функцией принадлежности

$\mu_Y = \{0.5/80, 0.6/85, 0.7/90, 0.8/95, 0.9/100, 1.0/105, 1.0/110\}$

Замечание. Здесь «высокий» и «тяжелый» - НМ разных ЛП, заданных на разных шкалах.

Тогда НО «высокий и тяжелый» на декартовом произведении X и Y будет представляться множеством пар {80/180, 85/180, … , 100/200}, а функцию принадлежности для точек (x,y) можно определить как **мягкое** произведение $\mu_{\text{«высокий и тяжелый»}} = \mu_X \cdot \mu_Y$

Пример 2. На универсальных множествах U=V={2, 3, 4} определены нечеткие множества X={1/2, 0.9/3, 0.6/4} и Y={1/3, 0.7/4}. Найти нечеткое множество декартового произведения XxY.

Решение:

XxY=min(1,1)/(2,3)+ min(0.9,0.7)/(2,4)+ min(0.9,1)/(3,3)+ min(0.9,0.7)/(3,4)+ min(0.6,1)/(4,3)+ +min(0.6,0.7)/(4,4)= 1/(2,3)+0.7/(2,4)+0.9/(3,3)+0.7/(3,4)+0.6/(4,3)+0.16/(4,4)

или

XxY={(1,(2,3)), (0.7,(2,4)), (0.9/(3,3)), (0.7,(3,4)), (0.6/(4,3)), (0.16,(4,4))}

Функция принадлежности нечеткого отношения с **непрерывным** носителем может быть задана некоторой функцией; например, для бинарного нечеткого отношения типа R:

$$\mu_R(u,v) = \begin{cases} 0, & u<v \\ f(u,v), & u \geq v \end{cases}$$

Заметим, что зачастую нечеткое отношение представляют матрицей функций предпочтений каждого отношения:

$$\mu_R = \begin{array}{c} \\ u_1 \\ \cdot \\ \cdot \\ \cdot \\ u_m \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & \ldots & v_n \end{array} \\ \begin{pmatrix} \mu_{11} & \ldots & \mu_{1n} \\ & \ldots & \\ \mu_{m1} & \ldots & \mu_{mn} \end{pmatrix} \end{array}$$

Пример. На множествах **U={7, 8, 9}** и **V={10, 15}** заданы нечеткие множества, соответственно, **X={1/7, 0.6/8, 0.3/9}** и **Y={1/10, 0.2/15}**. Найти бинарное нечеткое отношение (декартово произведение нечетких множеств). Решение:

R=X×Y={1/(7,10), 0.6/(8,10), 0.3/(9,10), 0.2/(7,15), 0.2/(8,15), 0.2/(9,15)}

С другой стороны, данное нечеткое бинарное отношение удобно представить матрицей, содержащей значения функции предпочтения для каждой пары:

$$\mu_R = \begin{array}{c} \\ 10 \\ 15 \end{array} \begin{array}{c} \begin{array}{ccc} 7 & 8 & 9 \end{array} \\ \begin{pmatrix} 1 & 0.6 & 0.3 \\ 0.2 & 0.2 & 0.2 \end{pmatrix} \end{array}$$

Иногда нечеткое отношение удобно изображать нечетким ориентированным графом, в котором элементы универсумов указываются в вершинах, а ребра характеризуют нечеткие отношения.

4.2.2. Операции над нечеткими отношениями

Носитель нечеткого бинарного отношения. Представляется множеством **R** упорядоченных пар **(u_k,v_j)**, каждая из которых имеет положительное значение функции принадлежности **{(u,v)| μ(u,v)>0}.**

1. Объединение R∪S: $\mu_{R\cup S}(u,v)=\max(\mu_R(u,v), \mu_S(u,v))$

Пример.

R	v_1	v_2	v_3
u_1	1	0.5	0.2
u_2	0.9	0.3	0

S	v_1	v_2	v_3
u_1	0.9	0.7	0.1
u_2	0.1	0.5	0.7

R∪S	v_1	v_2	v_3
u_1	1	0.7	0.2
u_2	0.9	0.5	0.7

2. Пересечение R∩S: $\mu_{R\cap S}(u,v)=\min(\mu_R(u,v), \mu_S(u,v))$

3. Дополнение R' : $\mu_{R'}(u,v)=1-\mu_R(u,v)$

4. Алгебраическое произведение R*S : $\mu_{R*S}(u,v)=\mu_R(u,v)\cdot\mu_S(u,v)$

5. Алгебраическая сумма

1). Собственно алгебраическая сумма R+S :

$\mu_{R+S}(u,v)=\mu_R(u,v)+\mu_S(u,v)-\mu_R(u,v)\cdot\mu_S(u,v)$

2). Дизъюнктивная сумма R⊕S: $R\oplus S=(R\cap S')\cup(R'\cap S)$

Пример. Пусть

R	v_1	v_2	v_3
u_1	1	0.5	0.2
u_2	0.9	0.3	0

S	v_1	v_2	v_3
u_1	0.9	0.7	0.1
u_2	0.1	0.5	0.7

(1) алгебраическая сумма:

R+S	v_1	v_2	v_3
u_1	1	0.85	0.28
u_2	0.91	0.65	0.7

(2) дизъюнктивная сумма

R	v_1	v_2	v_3
u_1	1	0.5	0.2
u_2	0.9	0.3	0

S	v_1	v_2	v_3
u_1	0.9	0.7	0.1
u_2	0.1	0.5	0.7

R’	v_1	v_2	v_3
u_1	0	0.5	0.8
u_2	0.1	0.7	1

S’	v_1	v_2	v_3
u_1	0.1	0.3	0.9
u_2	0.9	0.5	0.3

R∩S’	v_1	v_2	v_3
u_1	0.1	0.3	0.2
u_2	0.9	0.3	0

R’∩S	v_1	v_2	v_3
u_1	0	0.5	0.1
u_2	0.1	0.3	0.7

R⊕S	v_1	v_2	v_3
u_1	0.1	0.5	0.2
u_2	0.9	0.3	0.7

6. Проекция нечеткого отношения. Пусть **R** - нечеткое бинарное отношение в**U×V**. Тогда под **проекцией** (**тенью**) R на множество U понимают нечеткое множество $\mathbf{R_U}$, функция принадлежности которого определяется выражением

$\mu_{Ru}(u)=\max\mu_R(u,v)$, где максимум берется по $v\in V$.

Пример.

R	v_1	v_2	v_3
u_1	1	0.5	0.3
u_2	0.9	0.3	0

Тогда $\mathbf{R_U=\{1/ u_1, 0.9/u_2\}}$, $\mathbf{R_V=\{1/ v_1, 0.5/v_2, 0.3/v_3\}}$

7. Цилиндрическое продолжение проекции нечеткого отношения

Функции принадлежности цилиндрических продолжений **R1** и **R2** соответствующих проекций $\mathbf{R_U}$ и $\mathbf{R_V}$ определяются следующим образом:

$\mu_{R1}(u,v)=\mu_{Ru}(u)$, $\forall v\in V$

$\mu_{R2}(u,v)=\mu_{Rv}(v)$, $\forall u\in U$

Пример. Пусть множества **R**, **Ru** и **Rv** имеют вид:

$\mathbf{R=\{1/(u_1,v_1), 0,5/(u_1,v_2), 0.3/(u_1,v_3), 0.9/(u_2,v_1), 0.3/(u_2,v_2), 0/(u_2,v_3)\}}$

$\mathbf{R_U=\{1/u_1, 0.9/u_2\}}$

R_V={1/ v_1, 0.5/v_2, 0.3/v_3}
Тогда цилиндрические продолжения **R_1** и **R_2** будут иметь следующий вид:

R_1	v_1	v_2	v_3
u_1	1	1	1
u_2	0.9	0.9	0.9

R_2	v_1	v_2	v_3
u_1	1	0.5	0.5
u_2	1	0.5	0.3

8. Сепарабельность отношений
Если пересечение цилиндрических продолжений дает исходное отношение $R=R_1 \cap R_2$, оно называется **сепарабельным.** Для предыдущего примера
$R_1 \cap R_2$ = {1/(u_1,v_1), 0,5/(u_1,v_2), 0.3/(u_1,v_3), 0.9/(u_2,v_1), 0.5/(u_2,v_2), 0.3/(u_2,v_3)}, то есть, исходное отношение **R** не является сепарабельным.
9. Композиция нечетких отношений
Пусть заданы два нечетких бинарных отношения **R** и **S**, заданных, соответственно, на множествах **UxV** и **VxW**. Тогда функция принадлежности **композиции** нечетких отношений **R○S** будет определяться выражение

$$\mu_{R\circ S}(u,w) = \max_{\forall v \in V} \{\min(\mu_R(u,v), \mu_S(v,w)) | \forall u \in U, \forall w \in W\}$$

Пример. Вычислить композицию отношений, имеющих следующие функции принадлежности:

$$\mu_R = \begin{pmatrix} 0.2 & 0.7 \\ 0.6 & 0.8 \end{pmatrix} \qquad \mu_S = \begin{pmatrix} 0.5 & 0.8 \\ 0.4 & 1.0 \end{pmatrix}$$

Решение**.** Выполняя минимальную операцию, имеем:

$$\mu_{R\circ S} = \begin{pmatrix} \mu_{11} & \mu_{12} \\ \mu_{21} & \mu_{22} \end{pmatrix} = \begin{pmatrix} 0.2 & 0.7 \\ 0.6 & 0.8 \end{pmatrix} \begin{pmatrix} 0.5 & 0.8 \\ 0.4 & 1.0 \end{pmatrix} \qquad \begin{pmatrix} 0.4 & 0.7 \\ 0.5 & 0.8 \end{pmatrix}$$

Первый элемент произведения отношений получен согласно следующему соотношению:
μ_{11} = max {min(0.2, 0.5), min(0.7, 0.4)} = max{0.2, 0.4} = 0.4

4.3. Теория приближенных рассуждений

Под приближенными рассуждениями понимается процесс, при котором из нечетких посылок получают некоторые следствия, возможно, тоже нечеткие. Приближенные рассуждения лежат в основе способности человека понимать естественный язык, разбирать почерк, играть в игры, требующие умственных усилий, в общем, принимать решения в сложной и не полностью определенной среде. Эта способность рассуждений в качественных, неточных терминах отличает интеллект человека от интеллекта вычислительной машины.
Основным правилом вывода в традиционной логике является правило **modus ponens**, согласно которому мы судим об истинности высказывания **B** по истинности высказываний **A** и **A→B**.
Точнее, из истинности высказывания (**посылки**) "**Если** x есть P **то** y есть S" и истинности **события** “x есть P” справедливо **следствие** “y есть S”.

Если обозначить через **A** компоненту “x есть P”, а через **B** - “y есть S”, то импликация **A→B** будет соответствовать посылке.

Например, если **A**— высказывание "У студента высокая температура ", **B** — высказывание "Студент заболел", и истинны высказывания "У студента высокая температуры" и "Если температура высокая, то студент заболел", то истинно и высказывание "Студент заболел".

Во многих привычных рассуждениях, однако, правило **modus ponens** используется не в точной, а в приближенной форме. Так, обычно мы знаем, что **A** истинно и что $\mathbf{A^*}\mathbf{\rightarrow B}$.

где **A*** есть, в некотором смысле, приближение **A**. Тогда из $\mathbf{A^*}\mathbf{\rightarrow B}$ мы можем сделать вывод о том, что **B** приближенно истинно.

4.3.1. Композиционное правило вывода

Композиционное правило вывода — это всего лишь обобщение следующей знакомой процедуры. Предположим, что имеется кривая **y=f(x)** (рис. 19 (А)) и задано значение **x=a**. Тогда из того, что **y=f(x)** и **x=a** можно заключить, что **y=b=f(x)**.

Обобщим теперь этот процесс, предположив, что **a** — интервал, а **y=f(x)** — функция, значения которой суть интервалы, как на рис. 19 (Б). В этом случае, чтобы найти интервал **y=b**, соответствующий интервалу **a**, сначала построим цилиндрическое множество $\bar{\mathbf{a}}$ с основанием **a** и найдем его пересечение **I** с кривой, значения которой суть интервалы. Затем спроектируем это пересечение на ось **OY** и получим желаемое значение в виде интервала **b** (рис.21.).

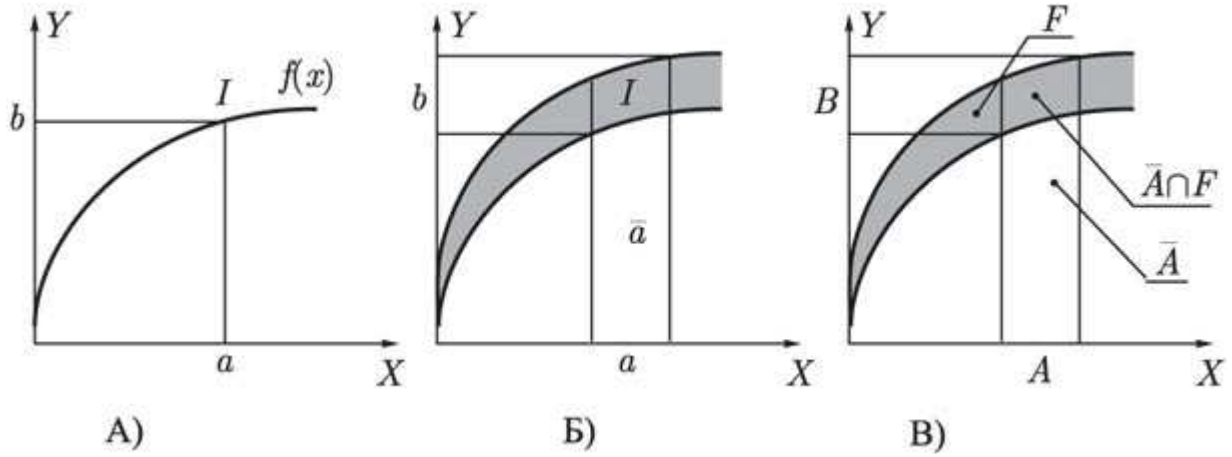


Рис.21. Графическая трактовка операции композиции нечетких отношений

Далее предположим, что **A** — нечеткое подмножество оси **OX** , а **F** — нечеткое отношение в **OX×OY** (см. рис.19(В)). Вновь образуя цилиндрическое нечеткое множество $\bar{\mathbf{A}}$ с основанием **A** и его пересечение с нечетким отношением **F**, мы получим нечеткое множество $\bar{\mathbf{A}} \cap \mathbf{F}$, которое является аналогом точки пересечения **I** на рис.19(А). Таким образом, из того, что **y=f(x)** и **x=A** — нечеткое подмножество оси **OX**, мы получаем значение **Y** в виде нечеткого подмножества **B** оси **OY**.

Правило. Пусть **U** и **V** — два универсальных множества с базовыми переменными **u** и **v**, соответственно. Пусть **A** и **F** — нечеткие подмножества множеств **U** и **U×V**.

Тогда композиционное правило вывода утверждает, что из нечетких множеств **A** и **F** следует нечеткое множество **B=A○F**.

Согласно определению композиции нечетких множеств, получим

$$\boldsymbol{\mu_B(v)} = \cup_{\boldsymbol{u \in U}} \left(\boldsymbol{\mu_A(u)} \cap \boldsymbol{\mu_F(u,v)}\right).$$

Пример. Пусть

U=V={1, 2, 3, 4}

A = **“МАЛЫЙ”**, μ_A**={1|1, 0.6|2, 0.2|3, 0|4}**

F=”ПРИМЕРНО РАВНЫ”=

	1	2	3	4
1	1	0,5	0	0
2	0,5	1	0,5	0
3	0	0,5	1	0,5
4	0	0	0,5	1

Тогда:

B={1|1, 0.6|2, 0.2|3, 0|4}$\circ$

	1	2	3	4
1	1	0,5	0	0
2	0,5	1	0,5	0
3	0	0,5	1	0,5
4	0	0	0,5	1

={1|1, 0.6|2, 0.5|3, 0.2|4},

что можно проинтерпретировать следующим образом:
B = “БОЛЕЕ ИЛИ МЕНЕЕ МАЛЫЙ”.
Словами этот приближенный вывод можно записать в виде

u – МАЛЫЙ **событие**
u, v – ПРИМЕРНО РАВНЫ **предпосылка**

v – БОЛЕЕ ИЛИ МЕНЕЕ МАЛЫЙ
приближенный вывод

4.3.2. Правило modus ponens как частный случай композиционного правила вывода

Правило **modus ponens** можно рассматривать как частный случай композиционного правила вывода. Чтобы установить эту связь, расширим понятие импликации на нечеткие множества. Пусть **A** и **B** — нечеткие высказывания (множества) и μ_A, μ_B — соответствующие им функции принадлежности. Тогда импликации **A→B** ет соответствовать некоторая функция принадлежности $\mu_{A\to B}$. По аналогии с традиционной логикой, можно предположить, что

$$A \to B \equiv \neg A \vee B$$

Тогда

$$\mu_{A\to B}(x,y) = \max\{1 - \mu_A(x), \mu_B(y)\}$$

Однако, это не единственное обобщение оператора импликации. Существуют различные интерпретации этого понятия.
Для дальнейшего рассмотрения ограничимся одним из них, принадлежащим Мамдани (Mamdani).
По Мамдани нечеткая импликация определяется следующим образом:

$$\mu_{A\to B}(x,y) = \min\{\mu_A(x), \mu_B(y)\}$$

Определим теперь обобщенное правило modus ponens:

Приведенная форм традиционной формулировки правила modus ponens :

Событие	A^*
Предпосылка	**A→B**
Вывод	$A^* \circ$**(A→B)**

1. допускается, гва,
2. **A*** необязате

4.3.3. Нечеткий логический вывод

Основой для проведения операции нечеткого логического вывода является база правил, содержащая нечеткие высказывания в форме "Если-то" и функции принадлежности для соответствующих лингвистических термов. При этом должны соблюдаться следующие условия:

1. Существует хотя бы одно правило для каждого лингвистического терма выходной переменной.
2. Для любого терма входной переменной имеется хотя бы одно правило, в котором этот терм используется в качестве предпосылки (левая часть правила).

В противном случае имеет место неполная база нечетких правил.

Пусть в базе правил имеется m правил вида:

R_1: ЕСЛИ x_1 это A_{11} … И … x_n это A_{1n}, ТО y это B_1
…
R_i: ЕСЛИ x_1 это A_{i1} … И … x_n это A_{in}, ТО y это B_i
…
R_m: ЕСЛИ x_1 это A_{i1} … И … x_n это A_{mn}, ТО y это B_m,

где x_k , k=1..n – входные переменные; y – выходная переменная; A_{ik}, B_i – заданные нечеткие множества с соответствующими функциями принадлежности.

Результатом нечеткого вывода является четкое значение переменной y^* на основе заданных четких значений x_k , k=1..n.

В общем случае механизм логического вывода включает четыре этапа: введение нечеткости (фазификация), нечеткий вывод, композиция и приведение к четкости, или дефазификация (см. рис. 22.).

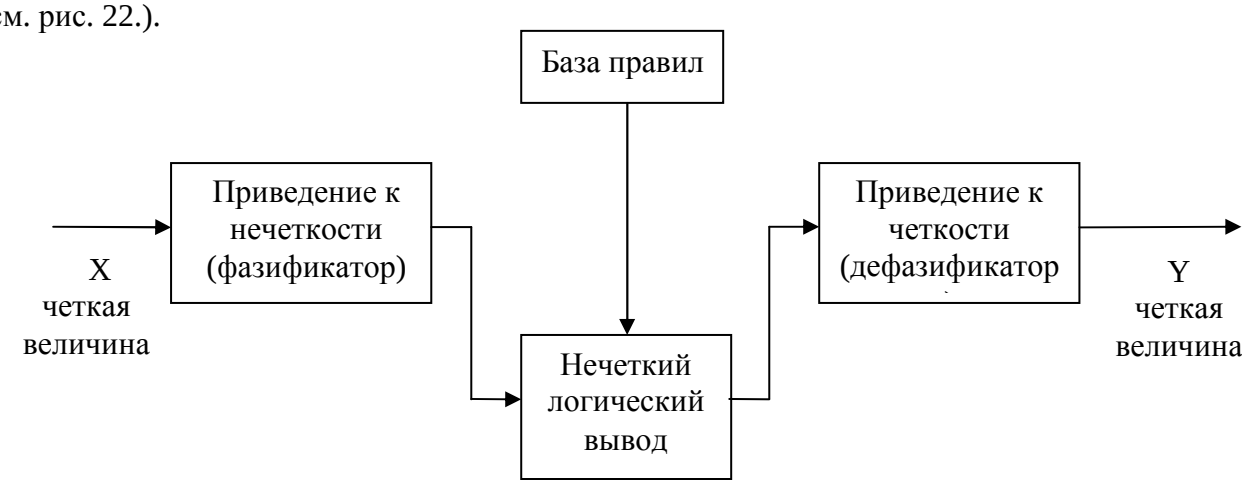


Рис.22. Система нечеткого логического вывода

Алгоритмы нечеткого вывода различаются главным образом видом используемых правил, логических операций и разновидностью метода дефазификации. Разработаны модели нечеткого вывода Мамдани, Сугено, Ларсена, Цукамото.

Рассмотрим подробнее нечеткий вывод на примере механизма Мамдани (Mamdani). Это наиболее распространенный способ логического вывода в нечетких системах. В нем используется минимаксная композиция нечетких множеств. Данный механизм включает в себя следующую последовательность действий.

1. **Фазификация**: определяются степени истинности, т.е. значения функций принадлежности для левых частей каждого правила (предпосылок). Для базы правил с m правилами обозначим степени истинности как $A_{ik}(x_k)$, i=1..m, k=1..n.
   Замечание. Здесь обозначение A_{ik} используется для краткости, заменяя стандартное представление функции предпочтения.
2. **Нечеткий вы**вод.
   1) Определяются уровни "отсечения" для левой части каждого из правил:

$$\alpha_i = \min_i (A_{ik}(x_k))$$

2) Находятся "усеченные" функции принадлежности:

$$B_i^*(y) = \min_i(\alpha_i, B_i(y))$$

3) **Композиция**, или объединение полученных усеченных функций:

$$MF(y) = \max_i(B_i^*(y))$$

где MF(y) – функция принадлежности итогового нечеткого множества.

3. **Дефазификация**, или приведение к четкости.
Существует несколько методов дефазификации. Например, метод среднего центра, или центроидный метод - геометрический смысл такого значения – центр тяжести для кривой MF(y).
Рис.23. графически показывает процесс нечеткого вывода по Мамдани для двух входных переменных и двух нечетких правил R1 и R2.

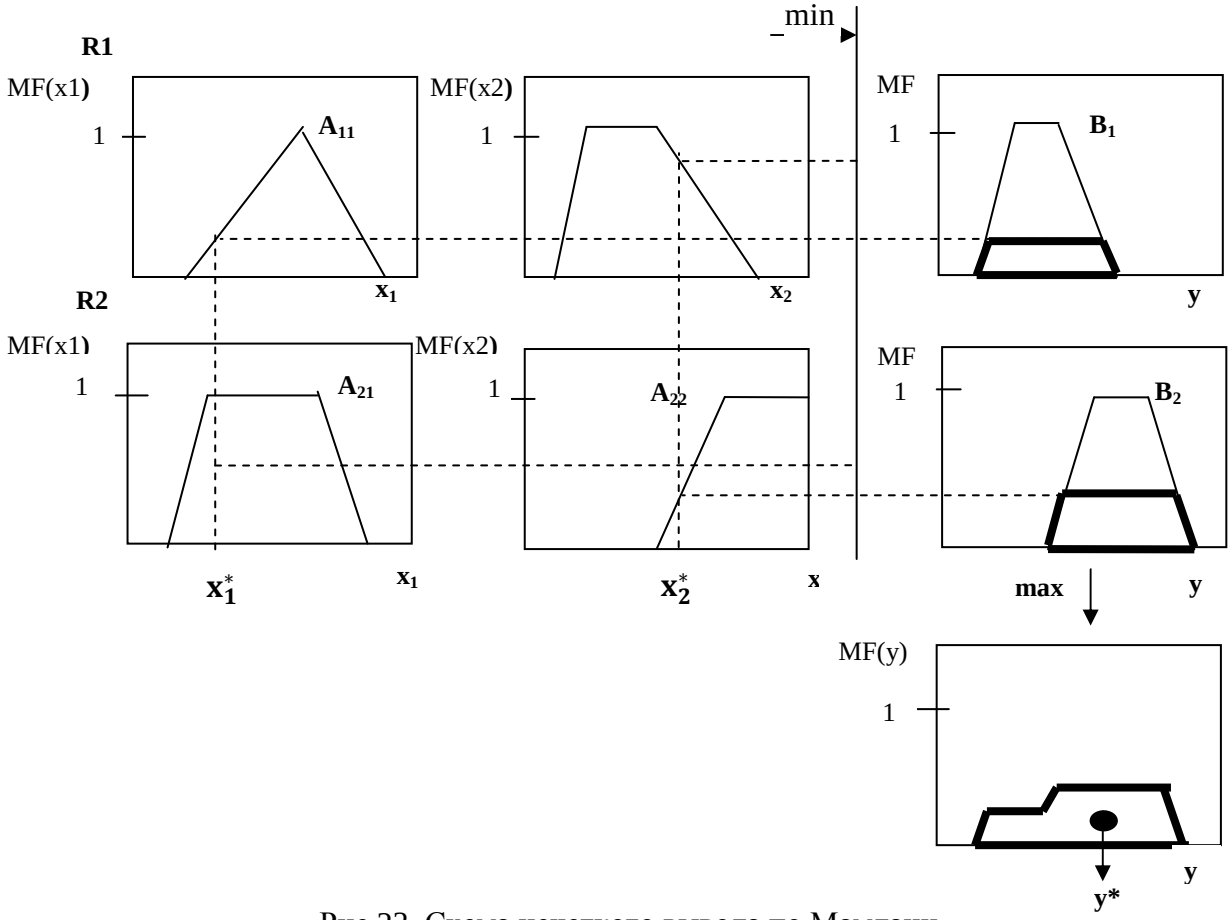


Рис.23. Схема нечеткого вывода по Мамдани

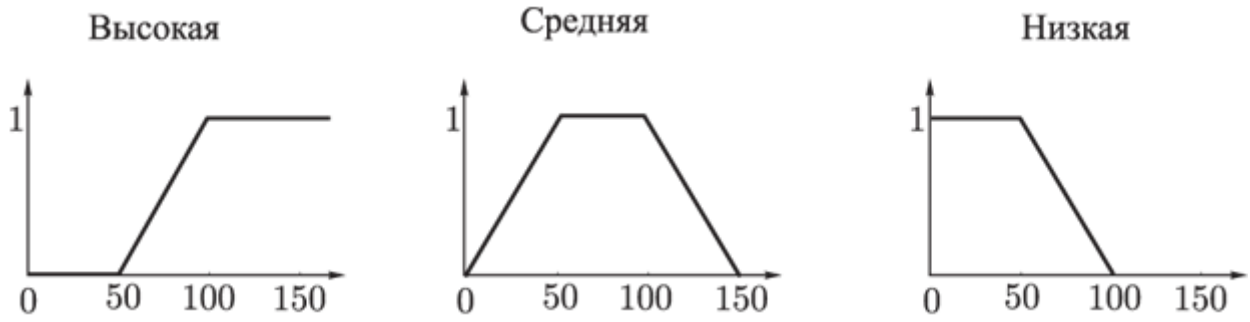
4.3.4. Примеры применения нечеткого вывода

Расчет параметров

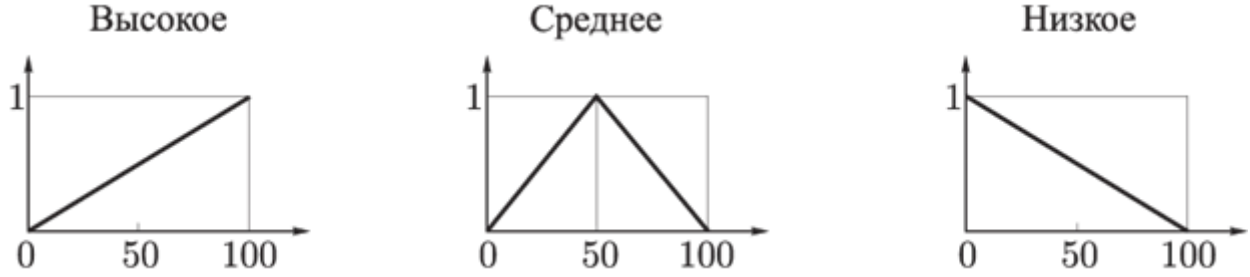
Пусть у нас есть некоторая система, например, реактор, описываемая тремя параметрами: температура, давление и расход рабочего вещества. Все показатели измеримы, и множество возможных значений известно. Также из опыта работы с системой известны некоторые правила, связывающие значения этих параметров. Предположим, что сломался датчик, измеряющий значение одного из параметров системы, но знать его показания необходимо хотя бы приблизительно. Тогда встает задача об отыскании этого неизвестного значения (пусть это будет давление) при известных показателях двух других параметров (температуры и расхода).

Введем в рассмотрение три лингвистические переменные — **Температуру**, **Давление** и **Расход**.

Температура. Множество термов (вербальных значений) - {**Высокая**, **Средняя**, **Низкая}**
Универсум (множество возможных значений) — отрезок **0 ÷ 150**.
Функции принадлежности термов имеют следующий вид:



Давление. Множество термов - {**Высокое, Среднее, Низкое**}.
Универсум — отрезок **0 ÷ 100.**
Функции принадлежности термов имеют следующий вид:



Расход. Начальное множество термов - {**Большой, Средний, Малый**}.
Универсум — отрезок **0 ÷ 8.**
Функции принадлежности термов имеют следующий вид:

База нечетких знаний, отражающая связи этих величин, представлена в виде следующих правил:

R1: **Если** Температура низкая **и** Расход малый, **то** Давление низкое;
R2: **Если** Температура средняя, **то** Давление среднее;
R3: **Если** Температура высокая **или** Расход большой, **то** Давление высокое.

Пусть известны значения **Температура = 85** и **Расход = 3,5** . Произведем расчет значения давления.
Этапы нечеткого вывода.
Этап 1. Фаззификация.
Получаем следующие степени уверенности:

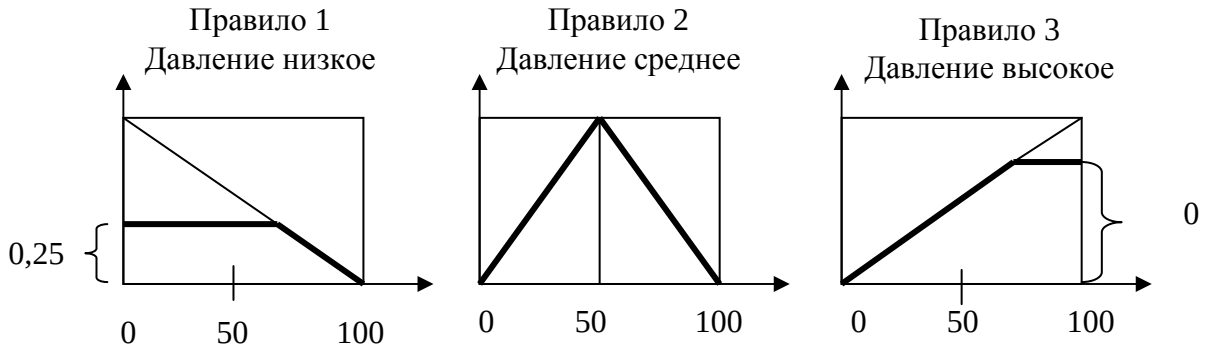
- Температура Высокая — 0,7;
- Температура Средняя — 1;
- Температура Низкая — 0,3;
- Расход Большой — 0;

- Расход Средний — 0,75;
- Расход Малый — 0,25.

Этап 2. Нечеткий вывод.

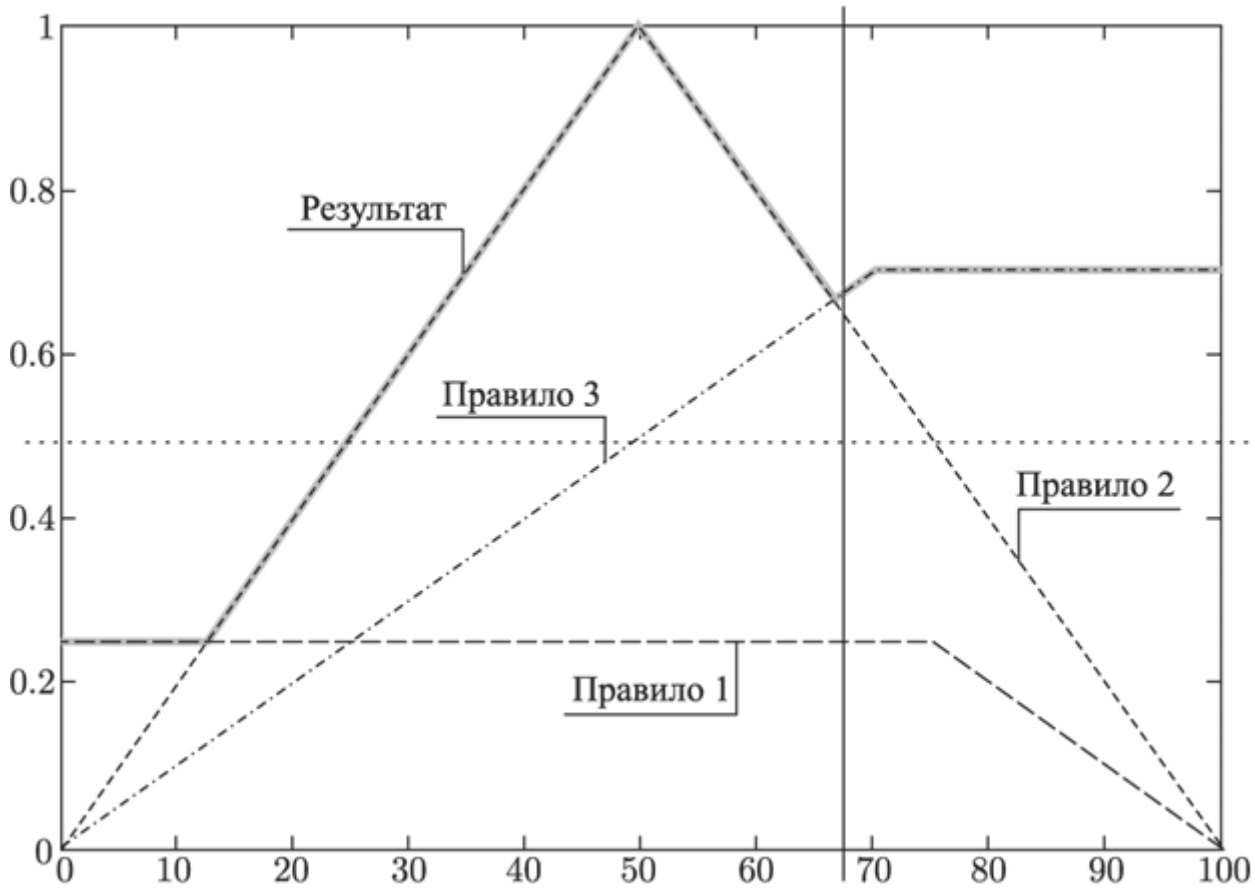
1) Вычисление степени уверенности посылок (правых частей) правил:
   R1: Температура низкая **и** Расход малый:
   min(Темп Низкая, Расход Малый) = min(0.3, 0.25) = **0.25**;
   R2: Температура Средняя: **1**;
   R3: Температура Высокая **или** Расход Большой:
   max(Темп Высокая, Расход Большой) = max(0.7,0) = **0,7**.

2) Используя определение нечеткой импликации по Мамдани - как минимума левой и правой частей, имеем:



3) Объединение результатов применения всех правил (**аккумуляция**).

Основной способов аккумуляции — построение максимума полученных функций принадлежности. Получаем:



Этап 3. Дефазификация
В данном случае достаточно метода первого максимума. Применяя его к полученной функции принадлежности, получаем, что значение давления — 50.
Таким образом, при температуре, равной 85, и расход рабочего вещества — 3,5, можно сделать вывод, что давление в реакторе примерно равно 50.

Нечеткие запросы к реляционным БД

Большая часть данных, обрабатываемых в современных информационных системах, носят четкий, числовой характер. Однако в запросах к базам данных, которые пытается формулировать человек, часто присутствуют неточности и неопределенности. Неудивительно, когда на запрос в поисковой системе Интернета пользователю выдается множество ссылок на документы, упорядоченных по степени релевантности (или соответствия) запросу. Потому что текстовой информации изначально присуща нечеткость и неопределенность, причинами которой является семантическая неоднозначность языка, наличие синонимов и т.д.
С базами данных информационных систем, или с четкими базами данных ситуация другая. Пусть, например, из базы данных требуется извлечь следующую информацию:

- "Получить список ***молодых*** сотрудников с ***невысокой*** заработной платой"
- "Найти предложения о сдаче не ***очень дорогого*** жилья ***близко*** к центру города"

Здесь высказывания "Молодой", "Невысокая", "Не очень дорогой", "Близко" имеют размытый, неточный характер, хотя заработная плата определена до рубля, а удаленность квартиры от центра – с точностью до километра. Причиной всему служит то, что в реальной жизни мы оперируем и рассуждаем неопределенными, неточными категориями. Такие запросы невозможно выполнить средствами языка SQL. И на помощь приходит концепция нечетких запросов.
Продемонстрируем ограниченность четких запросов на следующем примере. Пусть требуется получить сведения о менеджерах по продажам не старше 25 лет, у которых сумма годовых сделок превысила 200 тыс. по определенному региону, например, 1. Данный запрос можно записать на языке SQL следующим образом:

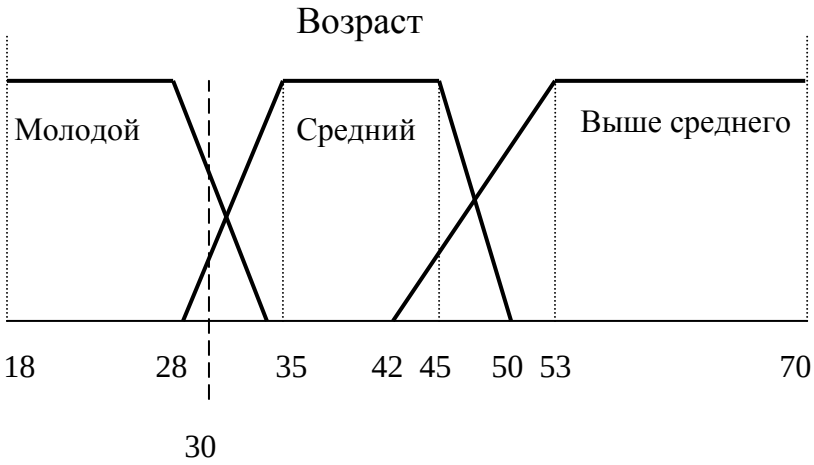
```
SELECT *  FROM Менеджеры
WHERE (Возраст <= 25 AND Сумма_продаж > 200000 AND Номер_региона = 1)
```

Менеджер по продажам 26 лет с годовой суммой продаж в 400 тыс., или 19 лет с суммой в 198 тыс. не попадут в результат запроса, хотя их характеристики почти удовлетворяют требованиям запроса.
Нечеткие запросы помогают справиться с подобными проблемами "пропадания" информации.
Нечеткие запросы помогают справиться с подобными проблемами "пропадания" информации.
Вернемся к примеру с менеджерами о продажах. Для простоты предположим, что вся необходимая информация находятся в одной таблице со следующими полями: ID – номер сотрудника, AGE – возраст и SUM (годовая сумма сделок).

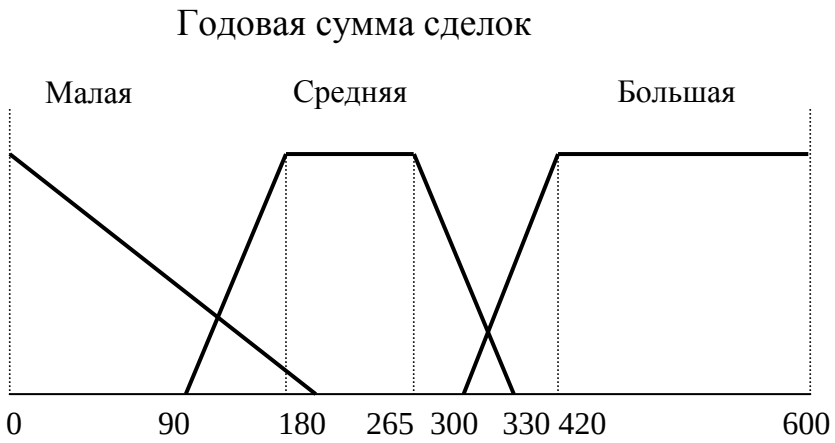
Менеджеры

Номер_региона	Возраст	Сумма_продаж
1	23	120 500
2	25	164 000
3	28	398 000
4	31	489 700
5	33	251 900
…	…	…

Для примера формализуем нечеткое понятие "Возраст сотрудника компании". Зададим для нее область определения X = [18; 70] и три лингвистических терма – "Молодой", "Средний", "Выше среднего".
Выберем трапецеидальные функции принадлежности со следующими координатами:
"Молодой" = [18, 18, 28, 34], "Средний" = [28, 35, 45, 50], "Выше среднего" = [42, 53, 60, 60].



Степень принадлежности сотрудника 30 лет к каждому из нечетких множеств:
MF[Молодой](30)=0,67; MF[Средний](30)=0,29; MF[Выше среднего](30)=0.
Определим еще одну лингвистическую переменную для поля SUM с областью определения X = [0; 600000] и термами "Малая", "Средняя" и "Большая" и аналогично построим для них функции принадлежности:
"Малая" = [0, 0, 0, 200000], "Средняя" = [90000, 180000, 265000, 330000], "Большая" = [300000, 420000, 600000, 600000].



Нечеткие запросы. Получить список всех ***молодых*** менеджеров по продажам с ***большой*** годовой суммой сделок, что на SQL-подобном синтаксисе запишется так:

SELECT * FROM Менеджер
WHERE (Возраст = "Молодой" AND Сумма_продаж = "Большая")
Рассчитав для каждой записи агрегированное значение функции принадлежности MF (при помощи операции нечетого "И"), получим результат нечеткого запроса:

Номер_региона	Возраст	Сумма продаж	Доля уверенности
3	28	398 000	0,82
4	31	489 700	0,50

Записи 1,2,5 не попали в результат запроса, т.к. для них значение функции принадлежности равно нулю. Записей, точно удовлетворяющих поставленному запросу (MF=1), в таблице не нашлось. Менеджер по продажам 28 лет и годовой суммой 398000 соответствует запросу с функцией принадлежности 0,82.
Аналогичный четкий запрос мог бы быть сформулирован, например, так:

SELECT * FROM Менеджеры WHERE (Возраст <= 28 AND Сумма_продаж >= 420000)

Его результат является пустым. Однако если мы немного расширим рамки возраста в запросе, то рискуем упустить других сотрудников с чуть более большим или меньшим возрастом. Поэтому можно сказать, что нечеткие запросы позволяют расширить область поиска в соответствии с изначально заданными человеком ограничениями.

Использование нечетких модификаторов

Для построения семантических конструкций, которые усиливают или ослабляют высказывания, например: "очень высокая цена", "приблизительно среднего возраста" и т.д., с использованием нечетких модификаторов генерируются новые лингвистические термы на основе базового терм-множества.
Нечеткие модификаторы усиливают или ослабляют высказывание. К усиливающим относится модификатор "Очень" (**Very**), к ослабляющим – "Более-или-менее" (**More-or-less**), или "Приблизительно", "Почти", нечеткие множества которых описываются функциями принадлежности вида:

$$\mu_{\mathbf{Very}}(x) = (\mu(x))^2$$
$$\mu_{\mathbf{More\text{-}or\text{-}less}}(x) = (\mu(x))^{0.5}$$

Используя нечеткие модификаторы, можно формировать более сложные запросы:

SELECT * FROM Менеджеры
WHERE (ВОЗРАСТ = "Более-или-менее Средний" AND СУММА_продаж = "Средняя")

Результат:

Номер_региона	Возраст	Сумма продаж	Доля уверенности
5	33	251 900	0,85

Нечеткие аналоги четких значений

Часто требуется оперировать не лингвистическими переменными, а нечеткими аналогами точных значений. Для этого существует нечеткое отношение "ОКОЛО" (Например, "Цена около 20"). Для реализации подобных нечетких отношений аналогично строится нечеткое множество с соответствующей функцией принадлежности, но уже на некотором относительном интервале (например, [-5; 5]), во избежании зависимости от контекста.
При вычислении функции принадлежности нечеткого отношения "Около Q" (Q – некоторое четкое число) производят масштабирование на относительный интервал.
Проиллюстрируем вышесказанное на примере таблицы с данными о ценных бумагах. Пусть она имеет в своем составе следующие поля: **Стоимость** (стоимость ценной бумаги), **Отношение** (отношение цены к прибыли), **Доход** (усредненный доход за последний квартал).

Таблица_бумаг

Номер_бумаги	Стоимость	Отношение	Доход
1	260	11	15,0
2	380	5	7,0
3	810	6	10,0
4	110	9	14,0
5	420	10	16,0

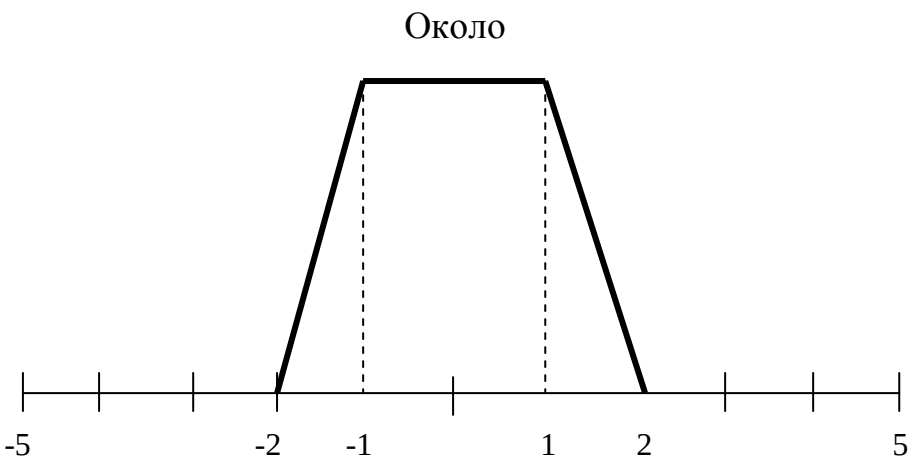
Пусть требуется найти ценные бумаги для покупки не дороже $150, с доходностью 15% и отношением цены прибыли 11. Это эквивалентно следующему SQL-запросу:

```
SELECT * FROM Таблица_бумаг
WHERE  ((Стоимость<=150) AND (Отношение=11) AND (Доход=15))
```

Результат такого запроса будет пустым.
Тогда сформулируем этот же запрос в нечетком виде с использованием отношения "ОКОЛО":

```
SELECT * FROM Таблица_бумаг WHERE  ((Стоимость =”Около 150”)
                                    AND (Отношение =”Около 11”)
                                    AND (Доход = ”Около 15”))
```

Построим нечеткое множество для отношения "ОКОЛО" в относительном интервале [-5; 5]. Это будет трапеция с координатами [-2, -1, 1, 2].



Рассчитаем значение нечеткого запроса "Цена около 250" для цены 380. Предварительно зададим области определения каждой лингвистической переменной:

- **Стоимость** – [0; 1000],
- **Отношение** – [0; 20],
- **Доход** – [0; 20].

Значение 130 (полученное как разница между 380 и 250) отмасштабируем на интервал [-5; 5], получим величину x=1,3 и MF(1,3)=0,7.
Применив нечеткое отношение **ОКОЛО** к каждому полю **Стоимость**, **Отношени** и **Доход** и рассчитав агрегированное значение функции принадлежности с помощью операции нечеткое "И", получим следующий результат запроса.

Номер_бумаги	Стоимость	Отношение	Доход	Доля_уверенности
1	260	11	15,0	1
4	110	9	14,0	0,9

Очевидным недостатком нечетких запросов является относительная субъективность функций принадлежности.