ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.В.ДВ.05.01 Представление знаний в интеллектуальных системах

Направление подготовки (специальность) 27.03.04 Управление в технических системах

Профиль подготовки (специализация) “Интеллектуальные системы обработки информации и управления”

Квалификация выпускника бакалавр

Форма обучения очная

СОДЕРЖАНИЕ

1. КОНСПЕКТ ЛЕКЦИЙ

1.1 Лекция № 1, 2 (4 часа).

Тема: «Знания и данные**»**

1.1.1 Краткое содержание вопросов:

При изучении интеллектуальных систем традиционно возникает вопрос - что же такое знания и чем они отличаются от обычных данных, десятилетиями обрабатываемых ЭВМ. Можно предложить несколько рабочих определений, в рамках которых это становится очевидным. Данные - это отдельные факты, характеризующие объекты, процессы и явления в предметной области, а также их свойства.

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

– данные как результат измерений и наблюдений;

– данные на материальных носителях информации (таблицы, протоколы, справочники);

– модели (структуры) данных в виде диаграмм, графиков, функций;

– данные в компьютере на языке описания данных;

– базы данных на машинных носителях.

Знания связаны с данными, основываются на них, но представляют результат мыслительной деятельности человека, обобщают его опыт, полученный в ходе выполнения какой-либо практической деятельности. Они получаются эмпирическим путем. Знания - это выявленные закономерности предметной области (принципы, связи, законы), позволяющие решать задачи в этой области.

При обработке на ЭВМ знания трансформируются аналогично данным:

– знания в памяти человека как результат мышления;

– материальные носители знаний (учебники, методические пособия);

– поле знаний - условное описание основных объектов предметной области, их атрибутов и закономерностей, их связывающих;

– знания, описанные на языках представления знаний (продукционные языки, семантические сети, фреймы - см. далее);

– базы знаний.

Часто используются такие определения знаний: знания - это хорошо структурированные данные, или данные о данных, или мета-данные. Для хранения данных используются базы данных (для них характерны большой объем и относительно небольшая удельная стоимость информации), для хранения знаний - базы знаний (небольшого объема, но исключительно дорогие информационные массивы). База знаний - основа любой интеллектуальной системы.

Модели представления знаний

Знания могут быть классифицированы по следующим категориям:

– *поверхностные* - знания о видимых взаимосвязях между отдельными событиями и фактами в предметной области;

– *глубинные* - абстракции, аналогии, схемы, отображающие структуру и процессы в предметной области.

Современные экспертные системы работают в основном с поверхностными знаниями. Это связано с тем, что на данный момент нет адекватных моделей, позволяющих работать с глубинными знаниями.

Кроме того, знания можно разделить на процедурные и декларативные. Исторически первичными были процедурные знания, т.е. знания, "растворенные" в

алгоритмах. Они управляли данными. Для их изменения требовалось изменять программы. Однако с развитием искусственного интеллекта приоритет данных постепенно изменялся, и все большая часть знаний сосредоточивалась в структурах данных (таблицы, списки, абстрактные типы данных), т.е. увеличивалась роль декларативных знаний.

Сегодня знания приобрели чисто декларативную форму, т.е. знаниями считаются предложения, записанные на языках представления знаний, приближенных к естественному и понятных неспециалистам.

Таким образом: Данные - это отдельные факты, характеризующие объекты и их свойства. Знания - это выявленные закономерности предметной области, представляют результат мыслительной деятельности человека. Знания могут быть классифицированы как поверхностные и глубинные, а также на процедурные и декларативные. Современные экспертные системы работают в основном с поверхностными и декларативными знаниями. Существуют десятки моделей и языков представления знаний для различных предметных областей.

1.2 Лекция № 3 (2 часа).

Тема: «Логика предикатов первого порядка**»**

1.2.1 Краткое содержание вопросов:

Логика предикатов, раздел современной символической логики, изучающий рассуждения и др. языковые контексты с учётом внутренней структуры входящих в них простых высказываний; при этом выражения языка трактуются функционально, т. е. как знаки некоторых функций или аргументов этих функций.

Важнейшая особенность Л. п. состоит в том, что т. н. общие имена (напр., «человек», «город», «металл»), знаки свойств («белый», «умный», «электропроводный») и знаки отношений («старше», «севернее», «тяжелее») рассматриваются как принадлежащие одной категории знаков, а именно категории предикатных символов, репрезентирующих функции, возможными аргументами которых являются объекты некоторого универсума рассмотрения, а значениями – истинностные оценки (в классич. логике – это «истина» и «ложь»). Предикатные символы различаются своей местностью: символы, представляющие предметно-истинностные функции от одного аргумента, называются одноместными, от двух аргументов – двухместными и т. д. (напр., предикатный символ «человек» одноместный, а «севернее» двухместный).

Другой отличит. чертой Л. п. является использование особого типа логич. символов – кванторов и связываемых ими (квантифицируемых) переменных для воспроизведения логич. форм множественных высказываний. Квантифицируемые переменные «пробегают» по множеству всех объектов рассмотрения, а роль квантора состоит в указании на ту часть объектов этого множества, для которых справедливо содержащееся в высказывании утверждение. Наиболее употребимы в логике квантор общности $\forall$

(в естеств. языке ему соответствуют термины типа «всякий», «каждый», «любой», «произвольный») и квантор существования $\exists$

(«существует», «найдётся», «имеется», «некоторый»).

Л. п. как раздел символич. логики включает в себя логич. теории разных типов, отличающиеся как выразительными возможностями языков, в которых они формулируются, так и классами выделяемых в них логических законов.

Различают Л. п. первого порядка и Л. п. высших порядков. В Л. п. первого порядка имеется лишь один тип квантифицируемых переменных – предметные (индивидные) переменные, возможными значениями которых являются индивиды (структура множественных высказываний воспроизводится здесь посредством формул вида $\forall xA$

– «Для всякого индивида x верно, что A», $\exists xA$ – «Существует индивид x, такой, что A»). В Л. п. второго порядка дополнительно вводятся переменные для различения признаков индивидов – их свойств и отношений между ними (эти переменные тоже разрешается связывать кванторами, получая выражения типа $\forall PA$ – «Для всякого свойства P верно, что A», $\exists RA$ – «Существует отношение R, такое, что A

»); в Л. п. третьего порядка разрешается квантификация по признакам признаков индивидов и т. д.

Выделяют также односортные и многосортные системы Л. п.: в односортной Л. п. все переменные, принадлежащие к одному и тому же типу, имеют одинаковую область пробега; в многосортной Л. п. с каждой переменной связывается собств. множество её возможных значений.

Л. п. включает как классич., так и неклассич. логич. теории. В основе классич. Л. п. лежат общие для всех классич. систем логики двузначности принцип и принцип экстенсиональности (значение сложного выражения зависит только от значений составляющих его выражений). Кроме того, в классич. Л. п. принимаются специфические именно для кванторной теории предпосылки экзистенциального характера – допущение о существовании объектов в предметной области и существовании денотатов у сингулярных (единичных) терминов. В неклассич. предикатных системах в той или иной форме происходит пересмотр указанных принципов. Напр., в свободной логике отказываются от обязат. существования индивидов в области интерпретации, а также допускается пустота единичных терминов.

Класс универсально общезначимых формул Л. п. первого порядка может быть формализован, т. е. существуют исчисления (синтаксически построенные логич. системы), классы теорем которых совпадают со множеством законов семантически построенной Л. п. Данный факт впервые установлен К. Гёделем (1930). Л. п. высших порядков являются принципиально неформализуемыми, т. е. нельзя построить адекватные им исчисления.

Среди др. метатеоретич. свойств Л. п. следует отметить её неразрешимость (отсутствие эффективной процедуры, позволяющей в конечное число шагов определять, является ли произвольная формула законом Л. п.), установленную А. Чёрчем (1936), и синтаксич. неполноту исчисления предикатов (т. е. возможность добавления в качестве новых аксиом некоторых недоказуемых формул без получения в системе противоречия). Последнее даёт возможность построения на базе Л. п. нетривиальных прикладных теорий. В этом случае вместо абстрактных предметных, предикатных и предметно-функциональных констант в алфавит вводятся конкретные термины словаря теории – имена объектов её предметной области, знаки их свойств и отношений, знаки заданных на данной области предметных функций. Сами прикладные теории первого порядка (их часто называют элементарными) строятся обычно аксиоматически: к логич. части (аксиомам и правилам вывода исчисления предикатов) добавляется собств. часть прикладной теории – постулаты, отражающие закономерности её предметной области. Простейшими примерами теорий первого порядка являются т. н. теории (логики) отношений: Л. п. с равенством, теория отношений эквивалентности, разл. теории порядка. Наиболее известная элементарная теория – система формальной арифметики Дж. Пеано.

1.3 Лекция № 4, 5 (4 часа).

Тема: «Язык программирования Пролог**»**

1.3.1 Краткое содержание вопросов:

Теоретические принципы Пролога

Пролог существенно отличается от языков, традиционно используемых в программировании. В языках Бейсик, Алгол и Паскаль основным методом программирования является разбиение задачи на дискретные шаги и их последовательное

описание. Последовательность шагов отображается в машинные команды, выполняемые компьютером. Отменить ранее выполненные команды невозможно, поскольку содержимое памяти постоянно обновляется. Языки программирования такого вида называются *алгоритмическими.* Алгоритм - математическая процедура, которая позволяет решить задачу за конечное число шагов. Пролог не относится к алгоритмическим языкам.

Свое название Пролог получил от слов «Программирование на языке ЛОГики». На самом деле Пролог не считается чистым языком логического программирования, но его создание - важный этап в этом направлении. Большой интерес к Прологу проявляют специалисты в области инженерии знаний и искусственного интеллекта. Растет его популярность в академическом мире.

Пролог был принят в качестве базового языка в японской программе создания ЭВМ пятого поколения, ориентированной на исследование методов логического программирования и искусственного интеллекта, а также на разработку нового поколения компьютеров, специально предназначенных для реализации данных методов.

При программировании на Прологе значительно упрощается описание решения, и программист имеет возможность заниматься непосредственно задачей, а не поиском способа разбиения ее решения на небольшие шаги, которые можно запрограммировать, как при программировании на алгоритмических языках.

Теоретической основой Пролога является раздел символьной логики, называемый *исчислением предикатов.* Прологу присущ ряд свойств, которыми не обладают традиционные языки программирования, что делает его мощным средством в области логического программирования. К таким свойствам относятся механизм вывода с поиском и возвратом, встроенный механизм сопоставления с образцом, и простая, но выразительная структура данных с возможностью ее изменения. Пролог отличает единообразие программ и данных. Данные и программы - лишь две различные точки зрения на объекты Пролога. В единой базе данных можно свободно создавать и уничтожать отдельные элементы. Поскольку не существует различия между программами и данными, можно менять программу во время ее работы. В Прологе отсутствуют указатели, операторы присваивания и GO__TO. Естественным методом программирования является рекурсия.

Многие интересные и полезные свойства Пролога непосредственно вытекают из его декларативности. Декларативность позволяет понимать программу, не отслеживая динамику ее выполнения, и является уникальной особенностью Пролога. Однако истинная мощь Пролога базируется не на одном каком-либо свойстве, а на сочетании всех его свойств. Мы приведем их подробное описание в последующих главах, а сейчас дадим общее представление о языке Пролог.

Мы не ставим перед собой цель описать исчисление предикатов, приведем лишь основные принципы, необходимые для изучения Пролога.

ПРОЛОГ является языком исчисления предикатов. Предикат – это логическая формула от одного или нескольких аргументов. Можно сказать, что предикат – это функция, отображающая множество произвольной природы в множество {ложно, истинно}. Обозначаются предикаты F(x), F(x, y), F(x, y, z) и т. д. Одноместный предикат F(x), определенный на предметной области M, является истинным, если у объекта x есть свойство F, и ложным – если этого свойства нет.

Двухместный предикат F(x, y) является истинным, если объекты x и y находятся в отношении F. Многоместный предикат F(x1 , x2 , x3 ,..., xN) задает отношение между элементами x1, x2, ..., xN и интерпретируется как обозначение высказывания: “Элементы x1 , x2, xN находятся между собой в отношении F”. При разработке логических программ предикаты получают обычно названия, соответствующие семантике описываемой предметной области.

Мы не будем употреблять термины «истинно» и «ложно», поскольку они определяют смысл результата. Доказанное утверждение обычно является истинным, но не

обязательно, так как доказательство зависит от известных фактов и сделанных на их основе выводов. Итак, мы ввели два новых термина. *Факт* есть некоторое утверждение, все факты определяются как доказанные. В частности, факт «рекс - это собака» считается доказанным по определению. Но такой запрос к факту, как «Является ли феликс собакой ?», не будет доказан с помощью известных фактов. Отрицательный ответ системы Пролог означает вовсе не то, что утверждение «феликс - собака» ложно, а лишь то, что у нас нет фактов о «феликсе» и о том, является ли он собакой. Другим термином, который мы использовали, был *вывод.* Имея множество фактов, мы можем определять новые свойства описанных в них объектов. Иными словами, допускается вывод свойств из фактов. Вернемся к факту «реке - это собака». Предположим, мы хотим на основе известных фактов выявить все объекты, обладающие свойством «быть собакой». Задав вопрос «Какие существа являются собаками?», получим ответ: «Рекс - это собака». Мы привели тривиальный пример вывода. Введем некоторую дополнительную информацию.

Если сформулировать приведенный выше запрос в алгоритмических терминах, то получится весьма сложное определение:

Объект является собакой в том случае, если он имеет свойство «быть собакой» или же имеет свойство «быть родителем» и существует объект, связанный с родителем и обладающий свойством «быть собакой».

Для того чтобы продемонстрировать алгоритмический подход к рассматриваемой задаче, приведем соответствующую программу на Паскале.

В приводимой ниже программе прежде всего описывается информация, относящаяся к «рексу», «голди» и «джоку», а затем перечисляются объекты, являющиеся собаками по определению и вследствие вывода (слайды 4 и 5).

Используемый нами синтаксис Пролога подробно будет определен позже. Пока примем, что каждый элемент, записанный строчными буквами, есть литерал. Он представляет объект и не может быть изменен. Каждое слово, начинающееся с прописной буквы (например, X, Y, Кто), является переменной. Во время выполнения программы переменным присваиваются значения. Символ ?- означает попытку согласовать, т.е. доказать следующую за ним цель (собака (Кто) является целью).

Мы определили, что рекс - это собака. Некоторый X является собакой (X называется *именованной переменной)* при условии, что X является родителем Y (другой свободной именованной переменной) и Y является собакой. Мы задали также условия, что голди - это родитель рекса и джок - это родитель рекса.

Синтаксис языка Турбо Пролог

Объекты данных Турбо Пролог

Система распознает тип объектов по его синтаксической форме в тексте программы. Это возможно благодаря тому, что синтаксис записи для различных типов имеет свои формы. Переменные начинаются с прописной буквы, атомы со строчной. Для изображения переменных и атомов используются буквы латинского алфавита: прописные (A, B, …, Z), строчные (a, b…z), цифры (0…9), специальные символы (, / (+ – * / < > = : _).

Атомы можно создать тремя способами:

1. из цепочки букв, цифр и символа подчеркивания, начиная со строчной буквы: x, sister, y10, x_y ;

2. из цепочки символов, заключенный в одинарные кавычки, это используется, если надо иметь атом начинающийся с прописной буквы: ' X ', ' Anna';

3. из специальных символов: < - - - >, = = = = . , . . . , :: =.

Объекты данных в Турбо-Прологе называются термами.

Константы представляют собой самоопределённые термы, т.е. их «значением» является их графическое представление (последовательность символов или число). Константы в Прологе используются для обозначения (именования) конкретных объектов предметной области и конкретных отношений между ними.

Переменная – последовательность букв, цифр и знака «подчеркивание», обязательно начинающаяся с прописной (большой) буквы или знака «подчеркивание». Среди переменных выделена переменная "_" (один знак подчеркивания), называемая анонимной. Она используется в ситуациях, когда нас не интересует её значение.

Область действия переменных

Областью действия переменной является утверждение. В пределах утверждения одно и то же имя принадлежит одной и той же переменной. Два утверждения могут использовать одно имя переменной совершенно различным образом. Единственным исключением из правила определения области действия переменных является анонимная переменная, например, "_" в цели любит(X,_). Каждая анонимная переменная есть отдельная сущность.

Она применяется тогда, когда конкретное значение переменной несущественно для данного утверждения. Таким образом, каждая анонимная переменная четко отличается от всех других анонимных переменных в утверждении. Переменные, отличные от анонимных, называются именованными, а неконкретизированные (переменные, которым не было присвоено значение) называются свободными.

Структура представляется на языке Пролог с помощью указания её функтора и компонент в следующем виде:

Имя структуры = функтор(компонента-1, ..., компонента-N),

где в качестве функтора должен выступать атом, а компонентой может быть любой терм (в том числе и структура).

Если объекты структуры относятся к одному и тому же типу доменов, то такая структура называется однодоменной. Если объекты структуры относятся к разным типам доменов – многодоменная.

Объекты могут быть константами, переменными или структурами. Для того, чтобы объединить объекты в структуры, надо выбрать функтор. Каждый функтор определяется двумя параметрами:

- именем, синтаксис которого совпадает с именем предиката;
- арностью – т. е. числом аргументов.

Рассмотрим в структуру date:

date (1, september, 2002),

где date – функтор, все аргументы данной многодоменной структуры являются константами.

date (Day, september,2002)

В этом примере date представляет функтор многодоменной структуры, аргументы которой являются переменные и константы.

likes ('Katja', fruits(banana, apples, oranges)),

А в этом примере likes – главный функтор, аргументы данной многодоменной структуры – константа и структура.

Рассмотренную структуру в программе можно описать следующим образом:

```
domains
personal_liking = fruits(type1,type2, type3)
type1, type2, type3 = symbol
predicates
likes(symbol, personal_liking)
```

Нормальная форма Бэкуса-Наура (БНФ),

Начнем с того, что познакомимся с так называемой **нормальной формой Бэкуса-Наура (БНФ)**, разработанной в 1960 Джоном Бэкусом и Питером Науром и используемой для формального описания синтаксиса языков программирования. Впервые БНФ была применена Питером Науром при записи синтаксиса языка Алгол-60.

При описании синтаксиса конструкций используются следующие обозначения:

Символ ::= читается как "по определению" ("это", "есть"). Слева от разделителя располагается объясняемое понятие, справа - конструкция, разъясняющая его. Например,

<Имя> ::= <Идентификатор>

В угловые скобки заключается часть выражения, которая используется для обозначения синтаксической конструкции языка, в частности объясняемое понятие. В приведенном выше примере это <Имя> и <Идентификатор>.

Символ | означает в нотации БНФ "или", он применяется для разделения различных альтернативных растолкований определяемого понятия.

Пример. Десятичную цифру можно определить следующим образом:

<цифра> ::= 0|1|2|3|4|5|6|7|8|9

Часть синтаксической конструкции, заключенная в квадратные скобки, является необязательной (может присутствовать или отсутствовать);

Пример. Запись

<Целое число> ::= [-]<Положительное целое число>

означает, что целое число можно определить через положительное целое число, перед которым может стоять знак минус.

Символ * обозначает, что часть синтаксической конструкции может повторяться произвольное число раз (ноль и более). Заметим, что иногда вместо символа * используют фигурные скобки ({,}).

Пример. Определить положительное целое число в нотации БНФ можно следующим образом:

<Положительное целое число> ::= <цифра>[<цифра>]*.

То есть положительное целое число состоит из одной или нескольких цифр.

Программа на языке Пролог, ее иногда называют базой знаний, состоит из предложений (или **утверждений**), каждое предложение заканчивается точкой.

Утверждения

Программирование на языке Пролог состоит из следующих этапов:

- Объявления некоторых фактов об объектах и отношениях между ними.
- Определения некоторых правил об объектах и отношениях между ними.
- Формулировки вопросов об объектах и отношений между ними.

Программа на Прологе есть совокупность утверждений. Утверждения состоят из целей и хранятся в базе данных Пролога. Таким образом, база данных Пролога может рассматриваться как программа на Прологе. В конце утверждения ставится точка ".". Иногда утверждение называется предложением. Основная операция Пролога - доказательство целей, входящих в утверждение.

Предложения бывают двух видов: *факты*, *правила*.

Предложение имеет вид

A:-

B1,... , Bn.

A называется заголовком или головой предложения, а B1,..., Bn - телом.

Факты

Факт констатирует, что между объектами выполнено некоторое отношение. Он состоит только из заголовка. Можно считать, что *факт* - это *предложение*, у которого тело пустое. Например, известный нам *факт*, что Наташа является мамой Даши, может быть записан в виде:

мама(Наташа, Даша).

Или предположим, что мы хотим сообщить Прологу факт: «Джону нравиться Мэри». Этот факт включает в себя два объекта Джон и Мэри и отношение между ними нравится. Этот факт на Прологе можно записать следующим образом:

нравится (джон, мэри)

Важно соблюдать следующие правила:

– Имена всех отношений и объектов должны начинаться со строчной буквы. Например, нравится, мэрии, джон.

– Сначала записывается имя отношения. Затем через запятую записываются имена объектов, а весь список имен объектов заключается в круглые скобки.

– Каждый факт должен заканчиваться точкой.

Факт представляет собой безусловно истинное утверждение.

В математической логике, отношения принято называть **предикатами.**

Если воспользоваться нормальной формой Бэкуса-Науэра, то предикат можно определить следующим образом:

<Предикат>::=<Имя> | <Имя>(<аргумент>[,<аргумент>]*),

т.е. предикат состоит либо только из имени, либо из имени и следующей за ним последовательности аргументов, заключенной в скобки. Аргументом или параметром предиката может быть константа, переменная или составной объект. Число аргументов предиката называется его **арностью** или **местностью**. В Турбо Прологе имя предиката должно состоять из последовательности латинских букв, цифр, знаков подчеркивания и начинаться с буквы или знака подчеркивания. В других версиях Пролога имя предиката может содержать символы не только из английского алфавита, но и из национального, например, из русского.

Соответственно, приведенный выше пример *факта* можно записать в Турбо Прологе, например, так:

mother("Наташа", "Даша").

Определяя с помощью фактов отношения между объектами, необходимо учитывать, в каком порядке перечисляются имена объектов внутри круглых скобок. Этот порядок может быть произвольным, но, выбрав один раз какой-то определенный порядок, мы должны следовать ему всегда.

Приведем некоторые факты и постараемся их интерпретировать:

ценный (золото)

женщина (джейн)

владеет (джон, золото)

отец (джон, мэри)

дает (джон, книга, мэрии)

Имена объектов и отношений являются полностью произвольным. Например, вместо терма нравится (джон, мэри) мы могли с таким же успехом написать a (b,c). Но для читабельности программы, лучше всего использовать имена, которые доносят до нас какой-то смысл.

Совокупность фактов в Прологе называется базой данных.

Некоторые предикаты уже известны системе, они называются **стандартными** или **встроенными.**

Унификация

Одним из наиболее важных аспектов программирования на Прологе являются понятия унификации (отождествления) и конкретизации переменных. Пролог пытается отождествить термы при доказательстве, или согласовании, целевого утверждения.

Например, в программе для согласования запроса ?- собака(X) целевое утверждение собака (X) было отождествлено с фактом собака (рекс), в результате чего переменная X стала конкретизированной: X= рекс. Переменные, входящие в утверждения, отождествляются особым образом - сопоставляются. Факт доказывается для всех значений переменной (переменных). Правило доказывается для всех значений переменных в головном целевом утверждении при условии, что

хвостовые целевые утверждения доказаны. Предполагается, что переменные в фактах и головных целевых утверждениях связаны квантором всеобщности. Переменные принимают конкретные значения на время доказательства целевого утверждения. В том случае, когда переменные содержатся только в хвостовых целевых утверждениях, правило считается доказанным, если хвостовое целевое утверждение истинно для одного или более значений переменных. Переменные, содержащиеся только в хвостовых целевых утверждениях, связаны квантором существования. Таким образом, они принимают конкретные значения на то время, когда целевое утверждение, в котором переменные были согласованы, остается доказанным. Терм X сопоставляется с термом Y по следующим правилам. Если X и Y - константы, то они сопоставимы, только если они одинаковы. Если X является константой или структурой, а Y - неконкретизированной переменной, то X и Y сопоставимы и Y принимает значение X (и наоборот). Если X и Y - структуры, то они сопоставимы тогда и только тогда, когда у них одни и те же главный функтор и арность и каждая из их соответствующих компонент сопоставима. Если X и Y - неконкретизированные (свободные) переменные, то они сопоставимы, в этом случае говорят, что они сцеплены.

1.4 Лекция № 6, 7 (4 часа).

Тема: «Правила-продукции»

1.4.1 Краткое содержание вопросов:

Самым распространенным форматом для представления знаний, наиболее соответствующим их процедурному характеру, является правило продукции, которое по своей сути - просто программа из одного оператора вида:

<ЕСЛИ условие, ТО действие>

Нотация процедур в виде последовательностей правил была впервые предложена математиком Постом.

Люди имеют запас "знаний" о мире, в котором они живут. Некоторые виды знаний общеизвестны; к ним относятся знания, связанные с приемом пищи или вождением автомобиля. Другие знания более специальные, например те, что используются экспертами. Знания обычно представляются в виде фактов, характерных для окружающего мира (т. е. классов объектов и взаимосвязей между ними), процедур и

правил манипулирования фактами, а также в виде информации о том, когда и как следует применять правила и процедуры.

Объекты группируют по классам, Петр, Джон, Фред и Анна могут мыслиться как объекты. Их можно отнести к классу “личность”. В дополнение к этому Петр, Джон и Фред могут быть классифицированы как “мужчины”, а Анна- как "женщина". Явное достоинство любой классификации заключается в том, что частично решается проблема переполнения памяти, так как достаточно помнить только характеристики класса, а не каждого объекта. Мы можем также определить отношения между классами (или отдельными объектами). Подобным образом мы можем определить отношение “руководит (А, В)”, означающее, что В находится в подчинении у А, В качестве примеров такой зависимости могут служить выражения:

руководит (Петр, Джон)

руководит (Джон, Анна)

руководит (Анна, Фред)

которые заключают в себе структуру “подотчетность” (другое отношение) между объектами Петр-Джон-Анна-Фред, Приведенный пример иллюстрирует отношения между схожими объектами. Между различающимися объектами также могут быть установлены отношения (например, “владеет (Петр, автомобиль)”). Знания об объектах и их взаимоотношениях позволяют классифицировать эти объекты и соотносить между собой.

Второй тип знания- правила. Они дают возможность определить, как вывести новые отличительные особенности класса или отношения для объектов, прежде не подразделенных на классы. Например, если мы определим отношение “отчитывается (В, А)” для того, чтобы отметить, что В подотчетен А (возможно, через других руководителей), то сможем установить правила:

“отчитывается (С, А)” есть ИСТИНА

ЕСЛИ или “руководит (А, С)” есть ИСТИНА

ИЛИ “руководит (А. В)” И “руководит (В, С)" есть ИСТИНА

Это довольно ограниченное правило. Оно применимо только для первого или второго уровня подотчетности, но в пределах ограничения оно дает нам возможность породить новый пример отношения “отчитывается”, который ранее не был известен. Например, правило позволяет сделать заключение о том, что “отчитывается (Анна, Петр)” и “отчитывается (Фред, Джон)” суть ИСТИНА. Вследствие того, что данное правило определено как двухуровневое, оно не может быть применено для получения вывода “отчитывается (Фред, Петр)” есть ИСТИНА. Для этого нам потребуется более мощное правило, включающее рекурсию:

“отчитывается (С, А)” есть ИСТИНА

ЕСЛИ или “руководит (А, С)” есть ИСТИНА

ИЛИ “руководит (А, В)” есть ИСТИНА

И “отчитывается (С, В)” есть ИСТИНА

Первая часть приведенного рекурсивного правила относится к прямой подотчетности, а вторая - к косвенной.

Если задать вопрос “отчитывается (Джон, Петр)?”, то ответом будет ИСТИНА, поскольку истинна первая часть правила “ЕСЛИ”. Вопрос “отчитывается (Фред, Петр)?” потребует более сложной обработки.

1.5 Лекция № 8 (2 часа).

Тема: «Семантические сети**»**

1.5.1 Краткое содержание вопросов:

Термин семантическая означает смысловая, а сама семантика — это наука, устанавливающая отношения между символами и объектами, которые они обозначают, т.е. наука, определяющая смысл знаков.

В основе семантических моделей лежит понятие сети, образованной помеченными вершинами и дугами. Вершины сети представляют некоторые сущности (абстрактные или конкретные объекты, процессы, события, явления), а дуги — отношения между сущностями, которые они связывают (связи типа «это», «имеет частью», «принадлежит» и т.п.). Наложив ограничения на описание дуг и вершин, можно получить сети различного вида. Если вершины не имеют собственной внутренней структуры, то соответствующие сети называются простыми. Если вершины обладают некоторой структурой, то такие сети называют иерархическими.

Характерной особенностью семантических сетей является обязательное наличие трех типов отношений: класс — элемент класса, свойство — значение и пример элемента класса.

В самом общем случае сетевая модель — это информационная модель предметной области. В сетевой модели представляются множество информационных единиц (объекты и их свойства, классы объектов и их свойств) и отношения между этими единицами.

В зависимости от типов отношений между информационными единицами различают сети:

классификационные, в которых используются отношения (типа часть – целое, род, вид, индивид), описывающие структуру предметной области, что позволяет отражать в базах знаний разные иерархические отношения между информационными единицами;

– функциональные (их часто называют вычислительными моделями), позволяющие описывать процедуры «вычислений» одних информационных единиц через другие;

– каузальные (называемые также сценариями), использующие причин-но-следственные отношения, а также отношения типа «средство — результат», «орудие — действие» и т.п.

– смешанные, использующие разнообразные типы отношений.

Если в сетевой модели допускаются отношения различного типа, то ее обычно называют семантической сетью.

Важной чертой семантических сетей является возможность представлять знания более естественным и структурированным образом, чем это делается с помощью других формализмов.

Виды отношений

При разработке представлений в виде семантических сетей важную роль играют следующие отношения:

– «является», отражающее принадлежность объекта некоторому классу объектов;

– «имеет», указывающее на то, что одно понятие представляет часть другого;

– «есть», указывающая на то, что одно понятие служит атрибутом другого.

Можно предложить несколько классификаций семантических сетей, связанных с типами отношений между понятиями. По количеству типов отношений сети делятся на однородные (с единственным типом отношений) инеоднородные (с различными типами отношений). По типам отношений их можно классифицировать какбинарные (в которых отношения связывают два объекта) in-арные (в которых есть специальные отношения, связывающие более двух понятий).

Семантические отношения бывают:

– Лингвистическими (глагольными, атрибутивными и падежными) — отображают смысловую взаимосвязь между событиями, между событиями и понятиями или свойствами. Они бывают глагольными, атрибутивными и падежными.

– Логическими — операциями, используемыми в алгебре логики (дизъюнкция, конъюнкция, инверсия и импликация).

– Теоретико-множественными — отношениями подмножества, части целого, множества и элемента.

– Квантифицированными — логическими кванторами общности и существования. Они используются для представления таких знаний как «Существует работник А, обслуживающий склад В» и т.д.

Наиболее часто в семантических сетях используются следующие отношения:

– связи типа «часть — целое» («класс — подкласс», «элемент — множество» и т.п.);

– функциональные связи (определяемые обычно глаголами «производит», «влияет» и т.п.);

– качественные связи (больше, меньше, равно и т.д.);

– пространственные связи (далеко от, близко от, за, под, над и т.п.);

– временные связи (раньше, позже, в течение и т.д.);

– атрибутивные связи (иметь свойство, иметь значение и т.п.);

– логические связи (и, или, не);

– и другие.

Проблема поиска решения в базе знаний типа семантической сети сводится к задаче поиска фрагмента сети, соответствующего некоторой подсети, соответствующей поставленному вопросу.

1.6 Лекция №9 (2 часа).

Тема: «Фреймы и объекты**»**

1.6.1 Краткое содержание вопросов:

Автором теории фреймов является профессор Массачусетского технологического института Мервин Минский, который в 70-егг. предложил фрейм как структуру знаний для восприятия пространственных сцен. В основе этой теории лежат психологические представления о том, как мы видим, слышим и концентрируем воспринимаемое. Сам Минский считал теорию фреймов скорее теорией постановки задач, чем продуктивной теорией, и суть ее излагал следующим образом. Каждый раз, попадая в некую ситуацию, человек вызывает из своей памяти соответствующую ситуации структуру, именуемую фреймом. Минский определил фрейм как единицу представления знания (абстрактного образа или ситуации), заполненную в прошлом, детали которой по необходимости изменяются и уточняются применительно к ситуации. Каждый фрейм может быть дополнен различной информацией, касающейся способов применения данного фрейма, последствий этого применения и т.п. Например, образ жизни каждого человека — это, большей частью, череда типовых ситуаций, различающихся каждый раз в деталях, но в общем и целом повторяющихся.

Фрейм — это структура знаний (т.е. декларативное представление), предназначенная для предоставления некоторой стандартной ситуации. С каждым фреймом ассоциируется разнообразная информация (в том числе и процедуры). Фрейм можно представить в виде сети, состоящей из вершин и отношений (дуг). Родственные фреймы связаны в систему фреймов. Система содержит описание зависимостей (причинных, временных и т.п.) между входящими в нее фреймами. Для выражения указанных зависимостей фреймы, входящие в систему, имеют общее множество слотов. Представление зависимостей в явном виде позволяет предсказать переход от одного состояния к другому, зависимому от него, состоянию и осуществить этот переход эффективно.

Для фреймовой модели характерно:

– представление знаний в виде достаточно крупных, содержательно завершенных единиц, называемых фреймами;

– иерархическая структура фреймов, где иерархия основана на степени абстрактности фреймов;

– совмещение во фреймах декларативных и процедурных знаний. Модель фрейма является достаточно универсальной, поскольку позволяет

– отобразить все многообразие знаний о мире через следующие виды фреймов:

– фреймы-структуры,для обозначения объектов и понятий (заем, залог,

– вексель);

– фреймы-роли (менеджер, кассир, клиент);

– фреймы-сценарии (банкротство, собрание акционеров, празднование

– именин);

– фреймы-ситуации (тревога, авария, рабочий режим устройства)

– и другие.

Фрейм имеет иерархическую структуру: на верхнем уровне располагаются фиксированные характеристики ситуации, на последующих уровнях (в так называемых «слотах» — отсеках) — уточняющая и конкретизирующая информация. Различают также пользовательское и машинное представление фреймов.

С точки зрения пользователя различают три уровня общности фреймов:

а) скелетный, пустой фрейм (шаблон), превращаемый после его заполнения в общее или конкретное понятие;

б) фрейм общего понятия (прототип или образец)— шаблон, заполненный не конкретными значениями, константами, а переменными;

в) фрейм конкретного понятия (экземпляр)— прототип, заполненный конкретными значениями, константами.

Фреймы-образцы хранятся в базе знаний, а фреймы-экземпляры создаются для отображения реальных фактических ситуаций на основе поступающих данных.

Структура фрейма может выглядеть следующим образом:

Ту же запись можно представить в виде таблицы, содержащей дополнительные столбцы для описания типа слота и возможного присоединения к тому или иному слоту специальных процедур, что допускается в теории фреймов.

Имя фрейма

имя слота	тип слота	значение слота	присоединенная процедура

Внутреннее (машинное) представление фрейма имеет более сложную организацию и содержит средства для создания иерархии фреймов, их взаимодействия, обмена

информацией, порождения конкретных фреймов из общих и общих фреймов из скелетных.

При конкретизации фрейма ему и его слотам присваиваются конкретные имена, а затем слоты заполняются значениями. Переход от исходного фреймапрототипа к фрейму-экземпляруможе быть многошаговым (за счет постепенного уточнения значений слотов).

Имя фрейма — это идентификатор, присваиваемый фрейму, уникальный во всей фреймовой системе. Фрейм состоит из произвольного числа слотов, часть из которых определяется системой.

Имя слота — это идентификатор слота, уникальный в пределах фрейма. Во фреймовых системах иерархического типа, основанных на отношениях «абстрактное — конкретное» создаются указатели наследования, которые

показывают, какую информацию об атрибутах слотов верхнего уровня наследуют слоты с такими же именами во фреймах нижнего уровня.

Указатель атрибутов слота — это указатель типа данных слота. К таким типам относятся FRAME (указатель); INTEGER (целое); REAL (вещественное); BOOL (булево); LISP (присоединенная процедура); TEXT (текст); LIST (список); TABLE (таблица); EXPRESSION (выражение) и др.

Каждый слот предназначен для заполнения определенной структурой данных (в скелетном фрейме все они пусты, кроме первого, который имеет значение).

Значение слота — значение, соответствующее типу данных слота и удовлетворяющее условиям наследования. Значением слота может быть практически все, что угодно: числа или математические соотношения, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма или других фреймов (сети фреймов).

У фреймовых моделей представления знаний нет специального механизма управления выводом и пользователь должен создавать его сам. Для этого используется три способа управления выводом: с помощью механизма наследования, демонов и присоединенных процедур.

Наследование свойств, заимствованное из теории семантических сетей, является важнейшим свойством теории фреймов. Во фреймах (как и в семантических сетях) наследование происходит поAKO-связям(A-Kind-Of = это). Слот AKO указывает на фрейм более высокого уровня иерархии, откуда неявно наследуются, т.е. переносятся значения аналогичных слотов.

Существуют несколько способов получения слотом значений во фрейме экземпляре:

- по умолчанию от фрейма образца (Default-значение);
- через наследование свойств от фрейма, указанного в слоте AKO;
- по формуле, указанной в слоте;
- через присоединенную процедуру;
- явно из диалога с пользователем;
- из баз данных.

В моделях представления знаний фреймами объединяются декларативные и процедурные знания. В качестве процедурных знаний выступают демоны и присоединенные процедуры.

Демон — процедура, автоматически запускаемая при обращении к слоту при выполнении некоторого условия:IF-NEEDED— «если нужно», запускается, если в момент обращения к слоту его значение не было установлено; IFADDED — «если добавлено», запускается при подстановке значения в слот; IFREMOVED — «если удалено», запускается при стирании значения слота. Демон — это разновидность присоединенной процедуры.

Присоединенная процедура — это служебная программа процедурного типа, используемая в качестве значения слота, запускается по сообщению, переданному из

другого фрейма (например, если поступает сообщение об изменении значения слота «зарплата» во фрейме ПЕТРОВ, то автоматически должна быть запущена процедура пересчета налога на Петрова во фрейме НАЛОГ).

Основным преимуществом фреймов как модели представления знаний является способность отражать концептуальную основу организации памяти человека, а также ее гибкость и наглядность.

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1 Практическое занятие №1 (2 часа).
Тема: «Знания и данные»

2.1.1 Краткое описание проводимого занятия:

Существует множество способов определять понятия. Один из широко применяемых способов основан на идее интенсионала. Интенсионал понятия - это определение через понятие более высокого уровня абстракции с указанием специфических свойств. Этот способ определяет знания. Другой способ определяет понятие через перечисление понятий более низкого уровня иерархии или фактов, относящихся к определяемому. Это есть определение через данные, или экстенсионал понятия.

Пример. Понятие "персональный компьютер". Его интенсионал: "Персональный компьютер - это дружественная ЭВМ, которую можно поставить на стол и купить менее чем за $2000". Экстенсионал этого понятия: "Персональный компьютер - это Mac, IBM PC, Sinkler...".

Существуют десятки моделей (или языков) представления знаний для различных предметных областей. Большинство из них может быть сведено к следующим классам:

- формальные логические модели;
- продукционные;
- семантические сети;
- фреймы.

2.2 Практическое занятие №2 (2 часа).
Тема: «Логика предикатов первого порядка»

2.2.1 Краткое описание проводимого занятия:

Язык классич. Л. п. первого порядка задаётся следующим образом. В качестве логич. символов вводятся некоторая функционально полная система пропозициональных связок (см. Логика высказываний) и кванторы. В алфавите содержится также бесконечный список предметных (индивидных) переменных. Среди нелогич. символов обязательно наличие непустого множества предикатных констант – аналога предикатных символов естеств. языка. Кроме того, в алфавит могут быть введены нелогич. символы других типов: предметные константы – аналоги собств. имён естеств. языка, а также предметно-функциональные константы разл. местности – аналоги предметных функторов (напр., «+», «возраст», «расстояние от... до...»). Технич. символами алфавита являются левая и правая скобки и запятая. В Л. п. имеется два типа правильно построенных выражений – термы (аналоги любых имён) и формулы (аналоги предложений).

Семантич. построение классической односортной Л. п. первого порядка может осуществляться разл. способами. Наиболее известны объектная и подстановочная семантики Л. п. Суть объектной семантики состоит в выборе некоторой непустой предметной области U

(универсума рассмотрения) и интерпретирующей функции I, сопоставляющей с нелогич. символами языка некие сущности (индивиды, предметно-истинностные и предметные функции), релятивизированные относительно U. Пару $<U,I>$ называют моделью или возможной реализацией. Далее задаются правила приписывания значений термам и формулам в модели $<U,I>$. При этом формула $\forall xA$ истинна в том случае, когда A истинно,

какой бы объект из U мы ни приписали в качестве значения переменной x (сохранив при этом значения остальных переменных), а $\exists xA$ истинна, если в универсуме найдётся такой объект, что при сопоставлении его в качестве значения переменной x формула A оказывается истинной. Законами Л. п. объявляются формулы, истинные в каждой модели $<U,I>$ (универсально общезначимые формулы). Смысл подстановочной семантики состоит в формулировке таких критериев истинности и ложности предложений языка, которые не предполагают соотнесения последних с внеязыковой действительностью, а опираются только на информацию о значениях элементарных формул. Здесь теория понимается как дедуктивно замкнутое множество предложений языка. Предложение вида $\forall xA$ ($\exists xA$) объявляется истинным в теории, если соответствующее бесквантворное утверждение справедливо для любого (хотя бы для одного) единичного термина, принадлежащего словарю данной теории.

2.3 Практическое занятие №3, 4 (4 часа).

Тема: «Язык программирования Пролог»

2.3.1 Краткое описание проводимого занятия:

В Турбо Прологе *предложения* с одним и тем же предикатом в заголовке должны идти одно за другим. Такая совокупность *предложений* называется **процедурой**. В приведенном выше примере про то, что Наташа является мамой Даши, мама - это имя двухаргументного предиката, у которого строковая константа "Наташа" является первым аргументом, а строковая константа "Даша" - вторым.

Правило - это *предложение*, истинность которого зависит от истинности одного или нескольких *предложений*. Обычно *правило* содержит несколько хвостовых *целей*, которые должны быть истинными для того, чтобы *правило* было истинным.

В нотации БНФ *правило* будет иметь вид:

<Правило>::=<предикат>:-<предикат>[,<предикат>]*.

Пример. Известно, что бабушка человека - это мама его мамы или мама его папы.

Соответствующие *правила* будут иметь вид:

```
бабушка(X,Y):-
мама(X,Z),мама(Z,Y).
бабушка(X,Y):-
мама(X,Z),папа(Z,Y).
```

Символ ":-" означает "если", и вместо него можно писать if.

Символ "," - это логическая связка "и" или конъюнкция, вместо него можно писать and.

Первое *правило* сообщает, что X является бабушкой Y, если существует такой Z, что X является мамой Z, а Z - мамой Y. Второе *правило* сообщает, что X является бабушкой Y, если существует такой Z, что X является мамой Z, а Z - папой Y.

В данном примере X, Y и Z - это переменные.

Переменная, которая получила какое-то значение и оказалась связанной с определенным объектом, называется **связанной**. Если переменная была конкретизирована каким-то значением и ей сопоставлен некоторый объект, то эта переменная уже не может быть изменена.

Областью действия переменной в Прологе является одно *предложение*. В разных *предложениях* может использоваться одно имя переменной для обозначения разных объектов. Исключением из правила определения области действия является **анонимная переменная**, которая обозначается символом подчеркивания "_". Анонимная переменная применяется в случае, когда значение переменной не важно. Каждая анонимная переменная - это отдельный объект.

Третьим специфическим видом *предложений* Пролога можно считать *вопросы*.

Имея некоторую совокупность фактов, мы можем обращаться к Прологу с вопросами о них. ***Вопрос*** состоит только из тела и может быть выражен с помощью БНФ в виде:

<Вопрос>::=<Предикат>[,<Предикат>]*

Вопросы используют для выяснения выполнимости некоторого отношения между описанными в программе объектами. Система рассматривает *вопрос* как *цель*, к которой надо стремиться. Ответ на *вопрос* может оказаться положительным или отрицательным, в зависимости от того, может ли быть достигнута соответствующая *цель*.

Программа на Прологе может содержать *вопрос* в программе (так называемая **внутренняя цель**). Если программа содержит внутреннюю *цель*, то после запуска программы на выполнение система проверяет достижимость заданной *цели*.

Если внутренней *цели* в программе нет, то после запуска программы система выдает приглашение вводить *вопросы* в диалоговом режиме (**внешняя цель**). Программа, компилируемая в исполняемый файл, обязательно должна иметь внутреннюю *цель*.

Если *цель* достигнута, система отвечает, что у нее есть информация, позволяющая сделать вывод об истинности *вопроса* (**"Yes"**). При этом если в *вопросе* содержатся переменные, то система либо выдает их значения, приводящие к решению, если решение существует, либо сообщает, что решений нет (**"No solution"**). Если достичь *цели* не удалось, система ответит, что у нее нет положительного ответа (**"No"**).

Следует заметить, что ответ **"No"** на *вопрос* не всегда означает, что отношение, о котором был задан *вопрос*, не выполняется. Система может дать такой ответ и в том случае, когда у нее просто нет информации, позволяющей положительно ответить на *вопрос*.

Можно сказать, что утверждение - это *правило*, а *факт* или *вопрос* - это его частный случай.

Рассмотрим несколько примеров. Пусть в программе заданы следующие отношения:

мама("Наташа","Даша").

мама("Даша","Маша").

Можно спросить у системы, является ли Наташа мамой Даши. Этот *вопрос* можно ввести в виде:

мама("Наташа","Даша")

Найдя соответствующий *факт* в программе, система ответит **"Yes"** (то есть **"Да"**). Если мы спросим:

мама("Наташа","Маша")

то получим ответ **"No"** (то есть **"Нет"**). Можно также попросить вывести имя мамы Даши:

мама(X,Даша).

Система сопоставит *вопрос* с первым *фактом*, конкретизирует переменную X значением "Наташа" и выдаст ответ:

X=Наташа

1 Solution

Вопрос об имени дочери Наташи записывается в виде:

мама(Наташа,X).

Соответствующим ответом будет:

X=Даша

1 Solution

Можно попросить систему найти имена всех известных ей мам и дочек, задав *вопрос*:

мама(X,Y).

Система последовательно будет пытаться согласовывать *вопрос* с имеющимися в программе *предложениями* от первого до последнего. В случае успешной унификации

соответствующих термов переменная X будет означена именем матери, а переменная Y - именем ее дочери.

В итоге получим ответ:

X=Наташа Y=Даша

X=Даша Y=Маша

2 solutions

Если надо получить только имена всех мам, можно воспользоваться анонимной переменной и записать *вопрос*:

мама(X,_).

Получим ответ:

X=Наташа

X=Даша

2 solutions

И, наконец, если надо получить ответ на *вопрос*: есть ли информация о людях, находящихся в отношении "мама - дочка", то его можно сформулировать в виде:

мама(_,_),

В данном случае нам не важны конкретные имена, а интересует, есть ли в нашей базе знаний хотя бы один соответствующий *факт*. Ответом в данном случае будет просто "**Yes**". Система сообщит о том, что у нее есть информация об объектах, связанных отношением "мама".

Введем в нашу программу *правило*, определяющее отношение "бабушка - внучка", в терминах "быть мамой":

бабушка(X,Y):-

мама(X,Z),

мама(Z,Y).

Заметим, что в нашей программе нет ни одного *факта*, связанного с отношением бабушка. Тем не менее, система оказывается способна найти ответы на *вопросы* о бабушках, пользуясь введенными *фактами* и *правилом*. Например, если нас интересует, чьей бабушкой является Наташа, то мы можем записать этот *вопрос* следующим образом:

бабушка("Наташа",X).

Для того чтобы найти ответ на *вопрос*, система просмотрит нашу базу сверху вниз, пытаясь найти *предложение*, в заголовке которого стоит предикат бабушка. Найдя такое *предложение* (это *предложение* бабушка(X,Y):-мама(X,Z),мама(Z,Y)), система конкретизирует переменную из заголовка *предложения* X именем "Наташа", переменную Y с переменной X из *вопроса*, после чего попытается достигнуть *цели*: мама("Наташа",Z) и мама(Z,Y). Для этого она просматривает базу знаний в поиске *предложения*, заголовок которого можно сопоставить с предикатом мама("Наташа",Z).

Это можно сделать, конкретизировав переменную Z именем "Даша". Затем система ищет *предложение*, в заголовке которого стоит предикат мама с первым аргументом "Даша" и каким-то именем в качестве второго аргумента. Подходящим *предложением* оказывается *факт* мама("Даша","Маша"). Система установила, что обе подцели мама("Наташа",Z) и мама(Z,Y) достижимы при Z="Даша", Y="Маша". Она выдает ответ:

X=Маша

Напомним, что наша переменная X из *вопроса* была связана с переменной Y из *правила*. После этого, если есть такая возможность, система пытается найти другие решения, удовлетворяющие *вопросу*. Однако в данном случае других решений нет.

Вообще говоря, *цель* может быть согласована, если она сопоставляется с заголовком какого-либо *предложения*. Если сопоставление происходит с *фактом*, то *цель* согласуется немедленно. Если же сопоставление происходит с заголовком *правила*, то *цель* согласуется только тогда, когда будет согласована каждая подцель в теле этого *правила*, после вызова ее в качестве *цели*. Подцели вызываются слева направо. Поиск подходящего для сопоставления *предложения* ведется с самого начала базы. Если подцель

не допускает сопоставления, то система совершает возврат для попытки повторного согласования подцели. При попытке повторного согласования система возобновляет просмотр базы с *предложения*, непосредственно следующего за тем, которое обеспечивало согласование *цели* ранее.

В программе на Прологе важен порядок *предложений* внутри процедуры, а также порядок хвостовых *целей* в теле *предложений*. От порядка *предложений* зависит порядок поиска решений и порядок, в котором будут находиться ответы на *вопросы*. Порядок *целей* влияет на количество проверок, выполняемых программой при решении.

При описании правил часто возникает необходимость использовать логические связки И и ИЛИ. В качестве связки И используется запятая, а в качестве связки ИЛИ – точка с запятой.

Например:
gigant(X) :- rost(X,Y),Y>200.
star_or_mlad(X) :- X>70; X<10.

ПРОЛОГ имеет большое количество встроенных предикатов, т.е. предикаты, определяемые автоматически. Например, встроенный предикат nl вызывает перевод строки, а встроенный предикат write применяется для вывода информации на экран. Встроенные предикаты используются так же, как и определяемые пользователем предикаты, но встроенный предикат не может являться головой правила или появляться в факте. Часто используемыми встроенными предикатами являются = (унификация) и логическое отрицание not.

Например:
student(X) :- X=”Петров”; X=”Иванов”.
xor_student(X) :- not(X=”Петров”), not(X=”Иванов”).
planeta(X) :- not(X=”солнце”).
Утверждение not(X = Y) эквивалентно X<>Y.

Иногда бывает полезно использовать предикаты, про которые заранее известно, истинны они или ложны. Для этих целей используют предикаты true и fail. Предикат true всегда истинен, в то время как fail всегда ложен. Последний предикат используется для управления процессом решения задачи на ПРОЛОГе.

ПРОЛОГ-программа может использовать комментарии, которые не влияют на выполнение программы, но могут оказать помощь человеку, читающему программу. ПРОЛОГ игнорирует произвольное число строк, заключенное между символами /* и */. Все, что находится между % и концом строки, также рассматривается как комментарий.

Сравнение с традиционными языками программирования.

Язык программирования характеризуется присущими ему механизмами управления и обработки данных. Пролог как универсальный язык программирования можно рассматривать и с этих точек зрения. Мы сравним средства управления последовательностью действий и обработку данных в Прологе и алголоподобных языках.

При успешном выполнении вычисления средства управления программ на языке Пролог подобны средствам управления в обычных процедурных языках. Обращение к некоторой цели соответствует вызову процедуры, порядок целей в теле правила соответствует последовательности операторов.

Точнее, правило A <- $B_1, B_2, \ldots, B_n$ можно рассматривать как определение процедуры:

procedure *A*

call *B1*,

call B_2

call *B,,*

end.

Рекурсивный вызов цели в Прологе в последовательности действий и в реализации подобен тому же вызову в обычных рекурсивных языках. Различие возникает при реализации возврата. В обычных языках, если вычисление не может быть продолжено (например, все ветви в операторе *case* ложны), возникает ошибка выполнения. В Прологе вычисление просто возвращается к последнему выбору и делается попытка продолжить вычисления по новому пути.

Структуры данных, которыми оперируют логические программы, - термы - соответствуют общим структурам записей в обычных языках программирования. Пролог использует очень гибкую систему организации структур данных. Подобно языку Лисп, Пролог является бестиповым языком, не содержащим объявления данных.

Другие особенности использования структур данных в языке Пролог связаны с природой логических переменных. Логические переменные соотносятся с объектами, а не с ячейками памяти. Если переменной сопоставлен конкретный объект, то эта переменная уже никогда не может ссылаться на другой объект. Иными словами, логическое программирование не поддерживает механизм деструктивного присваивания, позволяющий изменять значение инициализированной переменной.

В логическом программировании обработка данных полностью заключена в алгоритме унификации. В унификации реализованы:

- однократное присваивание,
- передача параметров,
- размещение записей,
- доступ к полям записей для одновременных чтения/записи.

2.4 Практическое занятие №5, 6 (4 часа).

Тема: «Правила-продукции»

2.4.1 Краткое описание проводимого занятия:

Объекты группируют по классам, Петр, Джон, Фред и Анна могут мыслиться как объекты. Их можно отнести к классу “личность”. В дополнение к этому Петр, Джон и Фред могут быть классифицированы как “мужчины”, а Анна- как "женщина". Явное достоинство любой классификации заключается в том, что частично решается проблема переполнения памяти, так как достаточно помнить только характеристики класса, а не каждого объекта. Мы можем также определить отношения между классами (или отдельными объектами). Подобным образом мы можем определить отношение “руководит (А, В)”, означающее, что В находится в подчинении у А, В качестве примеров такой зависимости могут служить выражения:

руководит (Петр, Джон)

руководит (Джон, Анна)

руководит (Анна, Фред)

которые заключают в себе структуру “подотчетность” (другое отношение) между объектами Петр-Джон-Анна-Фред, Приведенный пример иллюстрирует отношения между схожими объектами. Между различающимися объектами также могут быть установлены

отношения (например, “владеет (Петр, автомобиль)”). Знания об объектах и их взаимоотношениях позволяют классифицировать эти объекты и соотносить между собой.

Второй тип знания- правила. Они дают возможность определить, как вывести новые отличительные особенности класса или отношения для объектов, прежде не подразделенных на классы. Например, если мы определим отношение “отчитывается (B, A)” для того, чтобы отметить, что B подотчетен A (возможно, через других руководителей), то сможем установить правила:

“отчитывается (C, A)” есть ИСТИНА

ЕСЛИ или “руководит (A, C)” есть ИСТИНА

ИЛИ “руководит (A. B)” И “руководит (B, C)" есть ИСТИНА

Это довольно ограниченное правило. Оно применимо только для первого или второго уровня подотчетности, но в пределах ограничения оно дает нам возможность породить новый пример отношения “отчитывается”, который ранее не был известен. Например, правило позволяет сделать заключение о том, что “отчитывается (Анна, Петр)” и “отчитывается (Фред, Джон)” суть ИСТИНА. Вследствие того, что данное правило определено как двухуровневое, оно не может быть применено для получения вывода “отчитывается (Фред, Петр)” есть ИСТИНА. Для этого нам потребуется более мощное правило, включающее рекурсию:

“отчитывается (C, A)” есть ИСТИНА

ЕСЛИ или “руководит (A, C)” есть ИСТИНА

ИЛИ “руководит (A, B)” есть ИСТИНА

И “отчитывается (C, B)” есть ИСТИНА

Первая часть приведенного рекурсивного правила относится к прямой подотчетности, а вторая - к косвенной.

2.5 Практическое занятие №7 (2 часа).

Тема: «Семантические сети»

2.5.1 Краткое описание проводимого занятия:

При разработке представлений в виде семантических сетей важную роль играют следующие отношения:

- «является», отражающее принадлежность объекта некоторому классу объектов;
- «имеет», указывающее на то, что одно понятие представляет часть другого;
- «есть», указывающая на то, что одно понятие служит атрибутом другого.

Можно предложить несколько классификаций семантических сетей, связанных с типами отношений между понятиями. По количеству типов отношений сети делятся на однородные (с единственным типом отношений) инеоднородные (с различными типами отношений). По типам отношений их можно классифицировать какбинарные (в которых отношения связывают два объекта) in-арные (в которых есть специальные отношения, связывающие более двух понятий).

Семантические отношения бывают:

– Лингвистическими (глагольными, атрибутивными и падежными) — отображают смысловую взаимосвязь между событиями, между событиями и понятиями или свойствами. Они бывают глагольными, атрибутивными и падежными.

– Логическими — операциями, используемыми в алгебре логики (дизъюнкция, конъюнкция, инверсия и импликация).

– Теоретико-множественными — отношениями подмножества, части целого, множества и элемента.

– Квантифицированными — логическими кванторами общности и существования. Они используются для представления таких знаний как «Существует работник А, обслуживающий склад В» и т.д.

Наиболее часто в семантических сетях используются следующие отношения:

– связи типа «часть — целое» («класс — подкласс», «элемент — множество» и т.п.);

– функциональные связи (определяемые обычно глаголами «производит», «влияет» и т.п.);

– качественные связи (больше, меньше, равно и т.д.);

– пространственные связи (далеко от, близко от, за, под, над и т.п.);

– временные связи (раньше, позже, в течение и т.д.);

– атрибутивные связи (иметь свойство, иметь значение и т.п.);

– логические связи (и, или, не);

– и другие.

2.6 Практическое занятие №8 (2 часа).

Тема: «Фреймы и объекты»

2.6.1 Краткое описание проводимого занятия:

Структура фрейма может выглядеть следующим образом:

Ту же запись можно представить в виде таблицы, содержащей дополнительные столбцы для описания типа слота и возможного присоединения к тому или иному слоту специальных процедур, что допускается в теории фреймов.

Имя фрейма

имя слота	тип слота	значение слота	присоединенная процедура

Внутреннее (машинное) представление фрейма имеет более сложную организацию и содержит средства для создания иерархии фреймов, их взаимодействия, обмена информацией, порождения конкретных фреймов из общих и общих фреймов из скелетных.

При конкретизации фрейма ему и его слотам присваиваются конкретные имена, а затем слоты заполняются значениями. Переход от исходного фреймапрототипа к фрейму-экземпляруможет быть многошаговым (за счет постепенного уточнения значений слотов).

Имя фрейма — это идентификатор, присваиваемый фрейму, уникальный во всей фреймовой системе. Фрейм состоит из произвольного числа слотов, часть из которых определяется системой.

Имя слота — это идентификатор слота, уникальный в пределах фрейма. Во фреймовых системах иерархического типа, основанных на отношениях «абстрактное — конкретное» создаются указатели наследования, которые

показывают, какую информацию об атрибутах слотов верхнего уровня наследуют слоты с такими же именами во фреймах нижнего уровня.

Указатель атрибутов слота — это указатель типа данных слота. К таким типам относятся FRAME (указатель); INTEGER (целое); REAL (вещественное); BOOL (булево); LISP (присоединенная процедура); TEXT (текст); LIST (список); TABLE (таблица); EXPRESSION (выражение) и др.

Каждый слот предназначен для заполнения определенной структурой данных (в скелетном фрейме все они пусты, кроме первого, который имеет значение).

Значение слота — значение, соответствующее типу данных слота и удовлетворяющее условиям наследования. Значением слота может быть практически все, что угодно: числа или математические соотношения, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма или других фреймов (сети фреймов).

У фреймовых моделей представления знаний нет специального механизма управления выводом и пользователь должен создавать его сам. Для этого используется три способа управления выводом: с помощью механизма наследования, демонов и присоединенных процедур.

Наследование свойств, заимствованное из теории семантических сетей, является важнейшим свойством теории фреймов. Во фреймах (как и в семантических сетях) наследование происходит поAKO-связям(A-Kind-Of = это). Слот AKO указывает на фрейм более высокого уровня иерархии, откуда неявно наследуются, т.е. переносятся значения аналогичных слотов.

Существуют несколько способов получения слотом значений во фрейме экземпляре:

- по умолчанию от фрейма образца (Default-значение);
- через наследование свойств от фрейма, указанного в слоте AKO;
- по формуле, указанной в слоте;
- через присоединенную процедуру;
- явно из диалога с пользователем;
- из баз данных.