Федеральное агентство по образованию РФ
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ГОУВПО «АмГУ»)
Факультет математики и информатики

УТВЕРЖДАЮ
Зав. кафедрой ИиУС
_____А.В. Бушманов
«_____»_____2007 г

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ИНФОРМАЦИОННЫХ СИСТЕМАХ

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

для специальности 230201 «Информационные системы и технологии»

Составитель: Назаренко Н.В., ст. преподаватель кафедры ИиУС

Благовещенск 2007 г.

ББК 68.9 я 73
Н 75

Печатается по решению редакционно-издательского совета факультета математики и информатики Амурского государственного университета

Назаренко Н.В.

Учебно-методический комплекс по дисциплине «Представление знаний в информационных системах»: для студентов специальности 230201 «Информационные системы и технологии». – Благовещенск: Амурский гос. ун-т, 2007.

Учебно-методический комплекс разработан в соответствии с требованиями Государственного образовательного стандарта высшего профессионального образования специальности 230201 «Информационные системы и технологии» и ориентирован на оказание помощи студентам в изучении дисциплины «Представление знаний в информационных системах».

УМКД включает рабочую программу, календарно-тематический план дисциплины, кракий курс лекций, методические рекомендации по проведению и выполнению практических занятий, темы для самостоятельной работы, рекомендуемую литературу и вопросы к экзамену.

СОДЕРЖАНИЕ

I ПРИМЕРНАЯ ПРОГРАММА УЧЕБНОЙ ДИСЦИПЛИНЫ, УТВЕРЖДЕННАЯ МИНИСТЕРСТВОМ ОБРАЗОВАНИЯ РФ

Государственный образовательный стандарт высшего профессионального образования.

Направление подготовки дипломированного специалиста 654700 – Информационные системы.

Образовательная программа – 230102 Информационные системы и технологии.

Наименование дисциплины – Представление знаний в информационных системах.

Блок общепрофессиональных дисциплин, индекс ОПД.Ф.14.

Всего часов – 72

Логическая модель представления знаний и правила вывода; теоретические основы; пример спецификации и вычисления; продукционная модель представления знаний и правила их обработки; реляционные модели представления знаний и соответствующие способы рассуждений; фреймы, семантические сети; теория и техника приобретения знаний; принципы приобретения знаний. Существующие подходы и техника решения, экспертные системы - инструмент автоматизированных обучающих систем; введение в экспертные системы; роли эксперта, инженера знаний и пользователя; база знаний. Правила; объекты; определение запроса; редактор; процедурный язык; компилятор правил и объектов. Средства работы с файлами; структура главного меню; правила и объекты; антецедент и консеквент правила; первичная цель. Простые объекты; объекты со списком значений; объекты с фреймами; основные атрибуты (слоты) объекта; создание и редактирование процедур; вызов процедур из правил; процедурные фреймы и слоты; операторы процедурного языка; средства управления выполнением приложений; логическое программирование и экспертные системы; языки искусственного интеллекта; применение языка Пролог. Архитектура для автоматического рассуждения, основанного на правилах; механизм вывода на основе модели логического программирования; понятие о нечетких множествах и их связь с теорией построения экспертных систем; реализация экспертных систем в среде Windows.

II РАБОЧАЯ ПРОГРАММА

Курс 3 семестр 6
Зачет – 6 семестр

Лекции	18 (час.)
Лабораторные занятия	18 (час.)
Самостоятельная работа	36 (час.)
Всего часов	72 час.

1. Цели и задачи дисциплины, ее место в учебном процессе

1.1. Цели и задачи дисциплины

Цель дисциплины – получение теоретических знаний и практического опыта по представлению и обработке знаний, проектированию и использованию экспертных систем и баз знаний.

Задачи дисциплины:

дать основные понятия о моделях представления знаний в новых информационных технологиях, об алгоритмах логического вывода в различных моделях знаний, о подходах к построению моделей предметной области или выделенной сферы деятельности с точки зрения систем ИИ.

дать представление о новых технологиях обработки информации, основанных на знаниях, а также о системах, объединяемых понятием системы искусственного интеллекта.

1.2. Требования к уровню освоения содержания дисциплины

В результате изучения курса студенты должны:

знать основные понятия и направления исследований в области искусственного интеллекта, основные средства проектирования и разработки программ современными интеллектуальными методами, модели представления и получения знаний, методы инженерии знаний. Владеть языками логического и объектно-ориентированного программирования для решения интеллектуальных задач.

уметь применять необходимые методы искусственного интеллекта при разработке различных прикладных задач, строить модели простых предметных

областей в логических исчислениях высказываний и предикатов 1-го порядка, а также в продукционных и фреймовых представлениях; Практически использовать ПРОЛОГ, объектно-ориентированные и алгоритмические языки для разработки интеллектуальных задач.

1.3. **Перечень дисциплин с указанием разделов (тем), усвоение которых студентами необходимо при изучении данной дисциплины**

Изучение данной дисциплины требует от студентов предварительного усвоения таких дисциплин как «Информатика», «Информационные технологии», «Алгоритмические языки и программирование», «Технология программирования», «Управление данными» в объеме государственного образовательного стандарта высшего профессионального образования.

2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

2.1. Федеральный компонент

2.2. Наименование тем, их содержание, объем в лекционных часах

Дисциплина «Представление знаний в информационных системах» является дисциплиной, входящей в блок общепрофессиональных дисциплин федерального компонента для специальности 230201 «Информационные системы в технике и технологиях». Государственный стандарт – ОПД.Ф.14.

ТЕМАТИЧЕСКИЙ ПЛАН ЛЕКЦИОННЫХ ЗАНЯТИЙ

№ темы	Наименование темы	Кол-во часов
1	Информационный процесс представления знаний	2
2	Модели представления знаний и вывод на знаниях	4
3	Теория и техника приобретения знаний	4
4	Особенности разработки и использования экспертных систем.	4
5	Нечеткие знания и способы их обработки	2
6	Программный инструментарий разработки систем основанных на знаниях.	2
Итого		18

Тема 1: Информационный процесс представления знаний.

Этапы развития СИИ, эволюция представления знаний. Свойства и типы знаний. Декларативные и процедурные знания. Классификация систем основанных на знаниях. Типичные модели представления знаний. традиционные способы обработки знаний.

Тема 2: Модели представления знаний и вывод на знаниях

Логическая модель представления знаний и правила вывода; теоретические основы; пример спецификации и вычисления, продукционные модели представления знаний и правила их обработки, теоретическая модель продукционной системы (ПС). Структура ПС. Рабочая база данных. База правил. Механизм сопоставления. Достоинства и недостатки ПС. Реляционные модели представления знаний. Фрейм как структура представления знаний. Структура фрейма. Типы фреймов. Механизм присоединенных процедур, механизм наследования. Достоинства и недостатки фреймовых представлений. Семантические сети (СС). Виды СС: однородные, функциональнее, сценарные, иерархические. Основные типы отношений в СС. Механизм вывода на семантических сетях. Достоинства и недостатки СС.

Тема 3: Теория и техника приобретения знаний.

Принципы приобретения знаний. Теоретические аспекты извлечения знаний. Стратегии получения знаний. Структурирование знаний. Классификация методов практического извлечения знаний. Коммуникативные методы. Текстологические методы. Состояние и перспективы автоматизированного приобретения знаний.

Тема 4: Особенности разработки и использования экспертных систем.

Назначение и особенности методов искусственного интеллекта для разработки экспертных систем. Определение экспертной системы. Структура экспертных систем и основные понятия. Общие сведения об архитектуре экспертных систем. Пользовательский интерфейс, механизм вывода и база знаний. Классификация экспертных систем. Классификация инструментальных средств. Методология разработки экспертных систем. Этапы разработки экспертных систем. Основные направления в изучении и практическом использовании. Экспертные системы инструмент автоматизированных обучающих систем. Роли эксперта, инженера знаний и пользователя.

Тема 5: Нечеткие знания и способы их обработки

Понятие о нечетких множествах и их связь с теорией построения экспертных систем. Виды нечеткости знаний. Операции с нечеткими знаниями.

Тема 6: Программный инструментарий разработки систем основанных на знаниях.

Языки программирования для ИИ и языки представления знаний. Логическое программирование и экспертные системы. Применение языка Пролог.

2.3. Лабораторные занятия, их содержание и объем в часах.

ТЕМАТИЧЕСКИЙ ПЛАН ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Наименование темы	Кол-во часов
1. Работа с простейшими программами в системе Турбо-Пролог	2
2. Управление ходом выполнения программ в системе Турбо-Пролог	2
3. Создание отношений в Прологе	2
4. Рекурсия и рекурсивные процедуры в прологе	2
5. Списки и процедуры их обработки	4
6. Пролог-программы как простейшие базы данных и знаний	2
7. Способы представления баз данных в Пролог-программах	4
ИТОГО	18

2.4. Самостоятельная работа студентов (36 часов).

В качестве самостоятельной работы выполняется одно задание, которое состоит из 6-ти задач.

1. Построение модели предметной области для данной сферы деятельности человека.
2. Построение продукционной модели заданной предметной области в виде базы фактов (данных) и базы знаний и имитация работы этой модели в течение 5 тактов.
3. Построение фрагмента модели заданной предметной области на языке исчисления высказываний.
4. Построение фрагмента модели заданной предметной области на языке исчисления предикатов путём соответствующего усложнения.
5. Построение фрагмента модели предметной области в виде семантической сети.
6. Построение фрагмента предметной области в виде сети фреймов.

Темы заданий

Выбирается одна из приведенных ниже тем, и на ее основе выполняются все 6 вышеперечисленных задач. Указанные темы не являются обязательными. Студент вправе предложить свою, близкую ему, тему.

1. Сборка арки с помощью робота. Арка состоит из 2-х вертикальных и одной горизонтальных балок. Первоначально все балки находятся в различных точках стройплощадки.
2. Создание нового текстового файла в редакторе Word. Исходное состояние – компьютер выключен.
3. Запуск в рабочее состояние автомобиля. Начальное состояние – водитель рядом с машиной.

4. Заправка бензобака автомобиля у бензоколонки. Начальное состояние – автомобиль у бензоколонки.
5. Замена картриджа у принтера. Начальное состояние – картридж лежит рядом с принтером.
6. Прием–передача информации на пейджер с помощью пейджинговой компании. В процессе участвуют три человека: 1-й отправляет сообщение, 2-й принимает сообщение, 3-й – оператор – передает сообщение.
7. Установка автомашины в гараж. В начальный момент времени машина стоит у гаража, гараж закрыт.
8. Автоматизированный комплекс (конвейер) проверки наполнения бутылок квасом (водой, молоком и т.п.). В случае обнаружения недолива – доливает необходимое количество жидкости и поочередно снимает бутылки и ставит их в ящик.
9. Диагностика неисправностей блока питания компьютера.
10. Диагностика неисправностей компьютерного дисковода.
11. Ввод документа в компьютер с дискеты и его распечатка.
12. Передача документа по факсимильной связи.
13. Принятие решения о сертификации товара. Решение принимается на основе установления соответствия реквизитов товара (имя, № накладной, фирма, стоимость) реквизитам товаросопроводительных документов. При наличии несоответствий принимается решение об отправке на экспертизу. Если экспертиза подтверждает нарушение, то следует отказ в выдаче сертификата.
14. Диагностика неисправностей компьютера при его включении.
15. Управление лифтом (6 этажей, из двух или более нажатых кнопок выполняется та, которая меньше по номеру, если лифт идет вверх, и та, которая больше по номеру, если лифт идет вниз).
16. Заварка чая по-японски (по-русски, по-китайски и т.п.).
17. Приготовление борща.
18. Управление движением автомобилей на Т-образном перекрестке. Движение допускается в обе стороны. Решения: включить красный светофор, включить зеленый.
19. Управление потоком заданий на компьютер. Задания имеют ранг: 1, 2, 3; время решения соответственно: t1, t2, t3.
20. Обслуживание абонента в библиотеке.
21. Уход за комнатным цветком.
22. Обработка детали на станке, съём и замена детали.
23. Формирование каталога Интернет-ресурсов. Решение о регистрации принимается в случае соответствия заявленных данных о фирме поданным документам (тип деятельности, платежеспособность, оплата сервера и т.п.).
24. Диагностика неисправностей в работе принтера.
25. Форматирование дискеты и запись двух новых файлов.
26. Отправка посылки через почтовое отделение.

27. Ведение и обеспечение функционирования базы данных.
28. Ремонт копировального аппарата.
29. Осуществление модемной связи между двумя компьютерами. Предполагается, что один модем (ожидающий) ждет звонка, второй (инициирующий) осуществляет дозвон.
30. Рыбалка: смоделировать процесс рыбной ловли.
31. Сортировка поездов на сортировочной станции.
32. Уход за цветком в теплице.
33. Управление взлетом самолетов в аэропорту (одна взлетно-посадочная полоса).
34. Игра в "крестики-нолики".
35. Перевозка через реку волка, козы и капусты.
36. Распределение вакансий на бирже труда.
37. Заключение договора на поставку оборудования.

2.5. Вопросы к зачету

1. Основные направления исследований в ИИ.
2. Языки представления знаний.
3. Методы представления и использования знаний в ИИ.
4. Правила продукций.
5. Структура продукционной системы.
6. Нечеткая логика для продукционных систем.
7. Опишите на примере механизм работы базы правил.
8. Опишите механизм вывода в ПС с использованием прямой (обратной) стратегии. Приведите примеры.
9. Достоинства и недостатки продукционной модели.
10. Логика предикатов.
11. Правила выводов логики предикатов.
12. Метод резолюций в логике предикатов.
13. Семантические сети.
14. Фреймовые системы.
15. Типы фреймов.
16. Представление и использование нечетких знаний.
17. Операции с нечеткими знаниями.
18. Стратегии получения знаний.
19. Психологический аспект извлечения знаний.
20. Лингвистический аспект извлечения знаний
21. Гносеологический аспект извлечения знаний
22. Классификация методов извлечения знаний.
23. Активные методы извлечения знаний.
24. Пассивные методы извлечения знаний.
25. Текстологические методы извлечения знаний.
26. Методология структурирования знаний.
27. Стадии и методы структурирования знаний.

28. Состояние и перспективы автоматизированного приобретения знаний.
29. Эволюция систем приобретения знаний.
30. Классификация ЭС.
31. Структура и режимы экспертных систем.
32. Общие сведения об архитектуре ЭС.
33. Назначение и особенности методов ИИ для разработки ЭС.
34. Структура и режимы ЭС.
35. Характеристики ЭС.
36. Классификация инструментальных средств ЭС.
37. Методология разработки ЭС.
38. Этапы разработки ЭС.
39. Взаимодействия инженера по знаниям с экспертом.
40. Трудности разработки ЭС.
41. Проблемы и перспективы ЭС.
42. Возможности представления знаний на базе языка HTML.
43. Как работает механизм наследования?
44. Зачем нужны присоединенные процедуры?
45. Опишите механизм вывода на фреймах.
46. Укажите достоинства и недостатки фреймовых систем.
47. Дайте определение семантической сети.
48. Семантическая сеть с одним типом отношений.
49. Функции экспертных систем.
50. Роль базы знаний в экспертных системах.
51. Из каких функциональных блоков состоит ЭС?

2.5. Виды контроля.

Для проверки эффективности преподавания дисциплины проводится контроль знаний студентов. При этом используются следующие виды контроля:

- *текущий контроль* за аудиторной и самостоятельной работой обучаемых осуществляется во время проведения аудиторных занятий посредством устного опроса, проведения контрольных работ или осуществления лекции в форме диалога.
- *промежуточный контроль* осуществляется два раза в семестр в виде анализа итоговых отчетов на аттестационные вопросы.
- *итоговый контроль* в виде зачета осуществляется после успешного прохождения студентами текущего и промежуточного контроля и сдачи отчета по самостоятельной работе.

2.6. Требования к знаниям студентов, предъявляемые на зачете

Для получения зачета студент должен посещать занятия, проявлять активность в аудитории, обязан выполнить все лабораторные работы, знать теоретический материал в объеме лекционного курса.

3. УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ

3.1. Перечень обязательной (основной) литературы

1. Андрейчиков А.В., Андрейчикова О.Н. Интеллектуальные информационные системы: Учебник. – М.: Финансы и статистика, 2004. – 424 с.
2. Гаскаров Д.В. Интеллектуальные информационные системы. Учеб. Для вузов. – М.: Высш.шк., 2003. – 431 с.
3. Т.А. Гаврилова, В.Ф. Хорошевский Базы знаний интеллектуальных систем – СПб: Питер, 2000. – 384 с.
4. Питер Джексон Введение в экспертные системы.: Пер. с англ.: Уч. пос. – М.: Издательский дом «Вильямс», 2001. – 624 с.
5. Л.Стрейлинг, Э.Шапиро Искусство программирования на языке ПРОЛОГ.- М.:Мир, 1990.
6. Джордж Ф. Люгер Искусственный интеллект. Стратегии и методы решения сложных проблем. Изд-во: Вильямс, 2003, 864 с.

3.2. Перечень дополнительной литературы

1. А.Янсон Турбо-ПРОЛОГ в сжатом изложении, М.: Мир, 1998.
2. Ю.М.Смирнов Интеллектуализация ЭВМ. М.: Высшая школа, 1989.
3. К.Нейлор. Как построить свою экспертную систему. М.: Атомиздат, 1991.
4. Р.Форсайт Экспертные системы. М.: Радио и связь, 1987.
5. Гаврилова Т.А. Извлечение и структуирование знаний для ЭС, 1992
6. С. Осуга. Обработка знаний. М.: Мир, 1989.
7. "Представление и использование знаний" Х.Уэно,М.Исидзука, Москва "Мир" 1989
8. "Практическое введение в технологию искусственного интеллекта и экспертных систем с иллюстрациями на Бейсике", Р.Левин. Москва, "Финансы и статистика" 1990

3.3. Перечень методических пособий

1. Акилова И.М. Разработка экспертных систем. Практикум. Благовещенск, 2000 г.
2. Акилова И.М., Назаренко Н.В. Методы представления и обработки знаний. Практикум. Благовещенск, 2002. – 52 с.
3. Акилова И.М. Логическое программирование. Практикум. Благовещенск, 2002. – 40 с.

4. Необходимое техническое и программное обеспечение

Лекции проводятся в стандартной аудитории, оснащенной в соответствии с требованиями преподавания теоретических дисциплин.

Для проведения лабораторных работ необходим компьютерный класс на 12-14 посадочных рабочих мест пользователей. В классе должен быть установлен язык логического программирования PROLOG.

5. УЧЕБНО-МЕТОДИЧЕСКАЯ (ТЕХНОЛОГИЧЕСКАЯ) КАРТА ДИСЦИПЛИНЫ

Номер недели	Номер темы	Вопросы, изучаемые на лекции	Занятия (номера)		метод пособиянагляд. иИспользуемые	Самостоятельная работа студентов		Форма контроля
			(семин.)Практич.	Лаборат.		Содерж.	часы	
1	2	3	4	5	6	7	8	9
1	1			1	1, 3, 6 – осн. 1, 6 – доп 2, 3 – мет.			
3	2			2	1, 3, 6 – осн. 1, 6 – доп 2, 3 – мет			
5	2			3	1, 3, 6 – осн. 1, 6 – доп 2, 3 – мет			
7	3			4	1, 3, 6 – осн. 1, 6 – доп 2, 3 – мет			
9	3			5	1, 3, 6 – осн. 1, 6 – доп 2, 3 – мет			
11	4			5	1, 3, 6 – осн. 1, 6 – доп 2, 3 – мет			
13	4			6	1, 3, 6 – осн. 1, 6 – доп 2, 3 – мет			
15	5			7	1, 3, 6 – осн. 1, 6 – доп 2, 3 – мет			
17	6			7	1, 3, 6 – осн. 1, 6 – доп 2, 3 – мет			

Условные обозначения:
осн. – основная литература
доп. – дополнительная литература
мет. – методическое обеспечение

к.р. – контрольная работа
сб - собеседование

Приложение А
Образец тестовых заданий

1. Проблемами создания программ, с помощью которых можно решать те задачи, решение которых до этого считалось прерогативой человека, занимается:

а) бионическое направление;
б) программно-прагматическое направление;
в) нейрофизиологическое направление.

2. Что такое экстенсионал понятия:

а) набор близких по смыслу понятий;
б) расширение данного понятия;
в) набор конкретных фактов соответствующих данному понятию.

3. Получение инженером по знаниям наиболее полного из возможных представлений о предметной области и способах принятия решения в ней:

а) извлечение знаний;
б) приобретение знаний;
в) формирование знаний.

4. В семантической сети объектам, концепциям, событиям или понятиям обычно соответствуют:

а) отношения;
б) узлы;
в) дуги.

5. Метод представления знаний, в котором свойства описываются в терминах атрибутов и их значений и связываются с вершинами, представляющими концепции или объекты:

а) фрейм;
б) семантическая сеть;
в) логическая модель.

6. Раздел программы на ПРОЛОГе предназначенный для определения пользовательских типов данных:

а) domeins;
б) clauses;
в) domains.

7. Выполнение этой программы приведет к:

```
clauses
        road(«Cansas»,»Tampa»).
        road(«Tampa»,»Cansas»).
        road(«Tampa»,»Houston»).
goal:
        road(_,Y)
```

а) синтаксической ошибке;
б) присвоению атому Y значения «Tampa»;
в) получению списка («Tampa», («Cansas», «Houston».

8. Декларативный смысл Пролог-программы определяет:

а) что должно быть результатом программы;
б) как результат был достигнут;
в) как результат будет достигнут.

9. X является элементом списка, если в соответствии с первым предложением X – это голова списка, или в соответствии со вторым предложением X – это элемент хвоста:

а) member(X,[X| _]).
   member(X,[_| Y]):- member(X,Y).
б) member(X,[]).
   member(X,[X| Y]):- member(X,Y).
в) member(_,[_| Y]).
   member(X,[X| _]):- member(X,Y).

III ГРАФИК И МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ ПРОВЕДЕНИЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Содержание работы	**Количество часов**	**Сроки сдачи**	**Форма контроля**
1. Выбор темы	1	1 неделя	Письменный отчет в конце семестра
2. Построение модели предметной области (ПО)	3	2-3 неделя	
3. Построение продукционной модели предметной области	4	4-5 неделя	
4. Построение модели ПО на языке исчисления высказываний.	6	6-7 неделя	
5. Построение модели ПО на языке исчисления предикатов	6	8-9 неделя	
6. Построение модели ПО в виде семантической сети.	6	10-11 неделя	
7. Построение модели ПО в виде сети фреймов.	6	12-13неделя	
8. Оформление отчета	2	14 неделя	
Защита отчета	2	15 неделя	

Студенты выбирают одну из приведенных ниже тем, и на ее основе выполняют все 6 ниже перечисленных задач. Указанные темы не являются обязательными. Студент вправе предложить свою, близкую ему, тему.

Отчет по самостоятельной работе студенты сдают в письменном виде, оформленный с учетом требований к оформлению письменных работ. В конце работы необходимо привести перечень использованной литературы и других источников (ссылки на Internet и пр.).

В качестве самостоятельной работы выполняется одно задание, которое состоит из 6-ти задач.

1. Построение модели предметной области для данной сферы деятельности человека.

2. Построение продукционной модели заданной предметной области в виде базы фактов (данных) и базы знаний и имитация работы этой модели в течение 5 тактов.

3. Построение фрагмента модели заданной предметной области на языке исчисления высказываний.

4. Построение фрагмента модели заданной предметной области на языке исчисления предикатов путём соответствующего усложнения.

5. Построение фрагмента модели предметной области в виде семантической сети.

6. Построение фрагмента предметной области в виде сети фреймов.

Темы заданий

1. Сборка арки с помощью робота. Арка состоит из 2-х вертикальных и одной горизонтальных балок. Первоначально все балки находятся в различных точках стройплощадки.
2. Создание нового текстового файла в редакторе Word. Исходное состояние – компьютер выключен.
3. Запуск в рабочее состояние автомобиля. Начальное состояние – водитель рядом с машиной.
4. Заправка бензобака автомобиля у бензоколонки. Начальное состояние – автомобиль у бензоколонки.
5. Замена картриджа у принтера. Начальное состояние – картридж лежит рядом с принтером.
6. Прием–передача информации на пейджер с помощью пейджинговой компании. В процессе участвуют три человека: 1-й отправляет сообщение, 2-й принимает сообщение, 3-й – оператор – передает сообщение.
7. Установка автомашины в гараж. В начальный момент времени машина стоит у гаража, гараж закрыт.
8. Автоматизированный комплекс (конвейер) проверки наполнения бутылок квасом (водой, молоком и т.п.). В случае обнаружения недолива – доливает необходимое количество жидкости и поочередно снимает бутылки и ставит их в ящик.
9. Диагностика неисправностей блока питания компьютера.
10. Диагностика неисправностей компьютерного дисковода.
11. Ввод документа в компьютер с дискеты и его распечатка.
12. Передача документа по факсимильной связи.
13. Принятие решения о сертификации товара. Решение принимается на основе установления соответствия реквизитов товара (имя, № накладной, фирма, стоимость) реквизитам товаросопроводительных документов. При наличии несоответствий принимается решение об отправке на экспертизу. Если экспертиза подтверждает нарушение, то следует отказ в выдаче сертификата.
14. Диагностика неисправностей компьютера при его включении.
15. Управление лифтом (6 этажей, из двух или более нажатых кнопок выполняется та, которая меньше по номеру, если лифт идет вверх, и та, которая больше по номеру, если лифт идет вниз).

16. Заварка чая по-японски (по-русски, по-китайски и т.п.).
17. Приготовление борща.
18. Управление движением автомобилей на Т-образном перекрестке. Движение допускается в обе стороны. Решения: включить красный светофор, включить зеленый.
19. Управление потоком заданий на компьютер. Задания имеют ранг: 1, 2, 3; время решения соответственно: t1, t2, t3.
20. Обслуживание абонента в библиотеке.
21. Уход за комнатным цветком.
22. Обработка детали на станке, съём и замена детали.
23. Формирование каталога Интернет-ресурсов. Решение о регистрации принимается в случае соответствия заявленных данных о фирме поданным документам (тип деятельности, платежеспособность, оплата сервера и т.п.).
24. Диагностика неисправностей в работе принтера.
25. Форматирование дискеты и запись двух новых файлов.
26. Отправка посылки через почтовое отделение.
27. Ведение и обеспечение функционирования базы данных.
28. Ремонт копировального аппарата.
29. Осуществление модемной связи между двумя компьютерами. Предполагается, что один модем (ожидающий) ждет звонка, второй (инициирующий) осуществляет дозвон.
30. Рыбалка: смоделировать процесс рыбной ловли.
31. Сортировка поездов на сортировочной станции.
32. Уход за цветком в теплице.
33. Управление взлетом самолетов в аэропорту (одна взлетно-посадочная полоса).
34. Игра в "крестики-нолики".
35. Перевозка через реку волка, козы и капусты.
36. Распределение вакансий на бирже труда.
37. Заключение договора на поставку оборудования.

IV КРАТКИЙ КОНСПЕКТ ЛЕКЦИЙ

Тема 1: Информационный процесс представления знаний.

1. История развития систем искусственного интеллекта
2. Данные и знания
3. Классификация систем основанных на знаниях.

1.1. История развития систем искусственного интеллекта

Идея создания искусственного подобия человека для решения сложных задач и моделирования человеческого разума витала в воздухе еще в

древнейшие времена. Так, в древнем Египте была создана «оживающая» механическая статуя бога Амона. У Гомера в «Илиаде» бог Гефест ковал человекоподобные существа-автоматы. В литературе эта идея обыгрывалась многократно: от Галатеи Пигмалиона до Буратино папы Карло. Однако родоначальником искусственного интеллекта считается средневековый испанский философ, математик и поэт Раймонд Луллий, который еще в XIII веке попытался создать механическую машину для решения различных задач, на основе разработанной им всеобщей классификации понятий.

В XVIII веке Лейбниц и Декарт независимо друг от друга продолжили эту идею, предложив универсальные языки классификации всех наук. Эти работы можно считать первыми теоретическими работами в области искусственного интеллекта.

Окончательное рождение искусственного интеллекта как научного направления произошло только после создания ЭВМ в 40-х годах XX века. В это же время Нор-берт Винер создал свои основополагающие работы по новой науке кибернетике.

Термин «искусственный интеллект» ИИ (AI artificial intelligence) был предложен в 1956 т. на семинаре с аналогичным названием в Дартсмутском колледже (США). Семинар был посвящен разработке методов решения логических, а не вычислительных задач. В английском языке данное словосочетание не имеет той слегка фантастической антропоморфной окраски, которую оно приобрело в довольно неудачном русском переводе. Слово intelligence означает «умение рассуждать разумно», а вовсе не «интеллект», для которого есть термин intellect.

Вскоре после признания искусственного интеллекта отдельной областью науки произошло разделение его на два направления: нейрокибернетпика и «кибернетика черного ящика». Эти направления развиваются практически независимо, существенно различаясь как в методологии, так и в технологии. И только в настоящее время стали заметны тенденции к объединению этих частей вновь в единое целое.

1.2. Данные и знания

Приведем определения основных понятий изучаемой дисциплины и рассмотрим различие между понятиями «данные» и «знания».

Информация – совокупность сведений, воспринимаемых из окружающей среды, выдаваемых в окружающую среду либо сохраняемых внутри информационной системы.

Данные – представленная в формализованном виде конкретная информация об объектах предметной области, их свойствах и взаимосвязях, отражающая события и ситуации в этой области.

Данные представляются в виде, позволяющем автоматизировать их сбор, хранение и дальнейшую обработку человеком или информационным средством. Данные – это запись в соответствующем коде наблюдения, акта, объекта, песни, текста и т.д., пригодная для коммуникации, интерпретации, передачи, обработки и получения новой информации.

Информация получается из данных в результате решения некоторой задачи. Однако большая часть информации не может быть выведена из данных. Так, практически еще невозможен автоматический перевод художественных произведений с одного языка на другой. Трудно рассчитывать и на то, что в ближайшем будущем компьютер-переводчик сможет донести до нас тонкие оттенки юмора, чувств и т.п.

Информация, с которой имеет дело ЭВМ, разделяется на процедурную и декларативную. *Процедурная* информация представляется программами, которые выполняются в процессе решения задач, а *декларативная* – данными, которые обрабатывают эти программы.

Обычно как человек, так и ИС имеют дело с данными и знаниями из некоторой предметной области, например, математики, медицины, экономики и пр. Под *предметной областью* понимают информацию об объектах и связях между ними из некоторой области знаний.

Знание – это обобщенная и формализованная информация о свойствах и законах предметной области, с помощью которой реализуются процессы

решения задач, преобразования данных и самих знаний, и которая используется в процессе логического вывода.

Логический вывод – это генерирование новых утверждений (суждений) на основе исходных фактов, аксиом и правил вывода.

Знания с точки зрения решения задач в некоторой предметной области удобно разделить на две большие категории – факты и эвристики.

Под *фактами* обычно понимают общеизвестные в данной предметной области истинны, обстоятельства. *Эвристики* – это эмпирические алгоритмы, основанные на неформальных соображениях, которые ограничивают разнообразие и обеспечивают целенаправленность поведения решающей системы, не гарантируя, однако, получения наилучшего решения. Такие знания основываются на опыте специалиста (эксперта) в данной предметной области

Рассмотрим особенности знаний, в которых заключается их отличие от данных.

1. Интерпретация.
2. Структурированность.
3. Связность.
4. Семантическая метрика.
5. Активность.

Перечисленные пять особенностей определяют ту грань, за которой данные превращаются в знания, а БД – в БЗ. Совокупность средств, обеспечивающих работу со знаниями, образует СУБЗ.

1.3. Классификация систем основанных на знания.

Основным направлением, реализующим идеи ИИ, является разработка систем, основанных на знаниях. Центральный объект изучения ИИ – знания могут быть представлены в виде некоторой совокупности сведений (фактов, правил), процессов, явлений, а также способов решения задач данной предметной области. Специалисты, занимающиеся извлечением знаний, их формализацией и структурированием для обработки в компьютерных системах, называются *инженерами по знаниям* или *инженерами знаний*.

Структура систем, основанных на знаниях, может иметь следующий вид:

- извлечение знаний из различных источников;
- формирование качественных знаний;
- интеграция знаний;
- приобретение знаний от профессионалов;
- организация работы с экспертами;
- оценка и формализация знаний;
- модели знаний;
- логические системы;
- продукции;
- семантические сети;
- фреймы;
- системы представления знаний;
- базы знаний;
- манипулирование знаниями;
- пополнение знаний;
- классификация знаний;
- обобщение знаний;
- вывод на знаниях;
- методы резолюций;
- квазиаксиоматические системы;
- системы правдоподобного вывода;
- рассуждения с помощью знаний;
- объяснения на знаниях.

Для формализации знаний, представленных в текстовом, графическом виде, в виде документов и т.д., требуется наличие или, возможно, разработка методов, позволяющих преобразовать исходные знания к виду, пригодному для обработки в ИИС. Знания, полученные из различных источников, требуется интегрировать в связную и непротиворечивую систему.

Полученные от экспертов знания нужно оценить с точки зрения уже имеющихся в системе знаний, согласовать с последующими и выделить несовместные или противоречивые знания.

Для представления знаний используются логические модели, продукционные правила, таблицы принятия решений, семантические сети, фреймы и др. Для ввода, хранения, обработки, вывода знаний разработаны

системы управления базами знаний, которые включают языки описания и манипулирования знаниями, а также программные процедуры.

Особое место в манипулировании знаниями занимает вывод на знаниях, заключающийся в получении новых знаний на основе уже имеющихся в системе. Это одно из наиболее проблематичных с точки зрения реализации направлений в ИИС. Большой интерес в выводе знаний представляет манипулирование человеческими рассуждениями: аргументация на основе имеющихся знаний, рассуждения по аналогии и многое другое, чем люди пользуются в своей практике.

Тема 2: Модели представления знаний и вывод на знаниях

1. Логические модели
2. Продукционные модели
3. Семантические сети
4. Фреймы
5. Вывод на знаниях

Существуют десятки моделей (или языков) представления знаний для различных предметных областей. Большинство из них может быть сведено к следующим классам: логические модели, продукционные модели, семантические сети, фреймы.

2.1 Логические модели

Основная идея подхода при построении *логических моделей* представления знаний состоит в том, что вся информация, необходимая для решения прикладных задач, рассматривается как совокупность фактов и утверждений, которые представляются как формулы в некоторой логике. Знания отображаются совокупностью таких формул, а получение новых знаний сводится к реализации процедур логического вывода.

В основе логических моделей представления знаний лежит понятие формальной теории, задаваемое четверкой: S= <*B, F, A, R*>, где *B* — счетное множество базовых символов (алфавит), *F*— множество, называемое формулами, *A* — выделенное подмножество априори истинных формул (аксиом), *R* —

конечное множество отношений между формулами, называемое правилами вывода.

Достоинства логических моделей представления знаний:

1. В качестве «фундамента» здесь используется классический аппарат математической логики, методы которой достаточно хорошо изучены и формально обоснованы.

2. Существуют достаточно эффективные процедуры вывода, в том числе реализованные в языке логического программирования Пролог.

3. В базах знаний можно хранить лишь множество аксиом, а все остальные знания получать из них по правилам вывода.

Однако действительность не укладывается в рамки классической логики. Так называемая человеческая логика, применяемая при работе с неструктурированными знаниями, — это интеллектуальная модель с нечеткой структурой, и в этом ее отличие от «старой» (классической) логики. Таким образом, логики, адекватно отражающей человеческое мышление, к настоящему времени еще не создано.

2.2 Продукционные модели

Психологические исследования процессов принятия решений человеком показали, что рассуждая и принимая решения, человек использует правила продукций, или продукционные правила (от англ. *Production* — правило вывода, порождающее правило).

При использовании таких моделей у систем основанных на знаниях имеется возможность:

- применение простого и точного механизма использования знаний;
- представление знаний с высокой однородностью, описываемых по единому синтаксису.

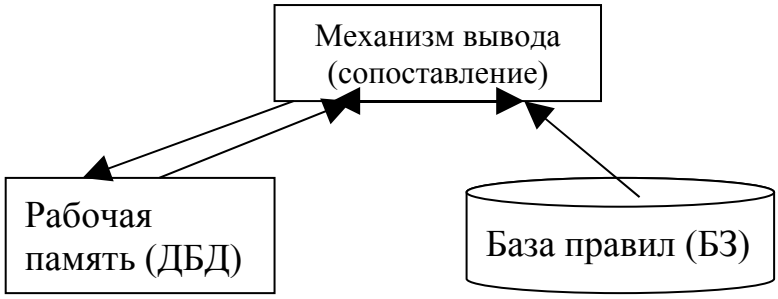
Эти две отличительные черты и определили широкое распространение методов представления знаний правилами. Впервые продукционные системы были предложены Постом в 1941 году. Продукция в системе Поста имеет следующую

схему: $\frac{t_1, t_2, \ldots t_n}{t}$, где $t_1,..t_n$ – посылки; t – заключение. Применение схемы Поста основывается на подстановке цепочек знаков вместо переменных, причем вместо вхождения одной и той же переменной представляется одна и та же цепочка.

Общим для систем продукций является то, что они состоят из трех элементов:

1. Набор правил, используемых как БЗ, его еще называют базой правил;
2. Рабочая память, где хранятся предпосылки, касающиеся отдельных задач, а также результаты выводов, получаемых на основе этих предпосылок (динамическая база данных ДБД);
3. Механизм логического вывода, использующий правила в соответствии с содержанием рабочей памяти.

Конфигурацию систем продукции упрощено можно представить в следующем виде:



В общем случае продукционное правило можно представить в следующем виде: *i: S; L; A-*B; Q,*

где *i* — индивидуальный номер продукции; S— описание класса ситуаций, в котором данная структура может использоваться; *L* — условие, при котором продукция активизируется; $A \rightarrow B$ — ядро продукции, например: «ЕСЛИ A_1, A_2 ..., A_n ТО *B»* . Такая запись означает, что «если все условия от A_1, до A_n являются истиной, то *B* также истина» или же «когда все условия от A_1, до A_n становятся истиной, то следует выполнить действие *B»; Q* — постусловие продукционного правила, описывает операции и действия (процедуры), которые необходимо выполнить после выполнения *B*. Например, внести изменения в данные либо в саму продукцию.

Суть использования правил продукции для представления знаний состоит в том, что левой части ставится в соответствие некоторое условие, а правой части — действие: ЕСЛИ **<перечень условия>,** ТО **<перечень действий>.** В такой интерпретации левая часть правил оценивается по отношению к базе данных (известному набору фактов) системы, и если эта оценка в определенном смысле соответствует логическому значению «ИСТИНА», то выполняется действие, заданное в правой части продукции.

В общем случае под условием (антецедентом) понимается некоторое предложение образец, по которому осуществляют поиск в базе знаний, а под действием (консеквентом) действия, выполняемые при успешном исходе поиска, это могут быть реальные действия, если система управляющая, или заключение вывод, представляющий собой новое (фактуальное) знание, или некоторая цель.

При использовании продукционной модели база знаний состоит из набора правил. Программа, управляющая перебором правил, называется *машиной вывода.* Механизм выводов связывает знания воедино, а затем выводит из последовательности знаний заключение.

В продукционных системах, основанных на знаниях, процесс обработки информации может осуществляться двумя способами. Первый предполагает обработку информации в прямом направлении (метод сопоставления). При втором подходе обработка информации осуществляется в обратном направлении метод «генерации» или выдвижения гипотезы и ее проверки (стратегия «от цели к данным»).

Таким образом, продукционные правила могут применяться к описанию состояния и описывать новые состояния (гипотезы) или же, напротив, использовать целевое состояние задачи как базу, когда система работает в обратном направлении. При этом продукционные правила применяются к целевому описанию для порождения подцелей (образуют систему редукций).

Свойства продукционных моделей:

1) Модульность

2) Каждое продукционное правило — самостоятельный элемент знаний
3) Простота интерпретации
4) Естественность

Недостатки продукционных систем проявляются тогда, когда число правил становится большим и возникают непредсказуемые побочные эффекты от изменения старого и добавления нового правила. Кроме того, затруднительна оценка целостного образа знаний, содержащихся в системе.

2.3 Семантические сети

Способ представления знаний с помощью *сетевых моделей* наиболее близок к тому, как они представлены в текстах на естественном языке. В его основе лежит идея о том, что вся необходимая информация может быть описана как совокупность троек *(arb),* где *a* и *b* объекты или понятия, а *r* бинарное отношение между ними.

Формально сетевые модели представления знаний могут быть заданы в виде $H = < I, C_1, \ldots C_n, Г>$, где I — множество информационных единиц, $C_1,..., C_n$ множество типов связей между элементами I, отображение Г задает между информационными единицами, входящими в I, связи из заданного набора типов связей $\{C_i.\}$.

В зависимости от типов связей $\{C_i.\}$ различают: классифицирующие сети, функциональные сети, сценарии.

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

Семантическая сеть — это модель, основой для которой является формализация знаний в виде ориентированного графа с размеченными вершинами и дугами. Вершинам соответствуют объекты, понятия или ситуации, а дугам — отношения между ними. В качестве понятий обычно выступают абстрактные или конкретные объекты, а отношения – это связи типа: «это» («АКО» – A-Kind-Of», «is»), «имеет частью» («has part»), «принадлежит», «любит». Это наиболее общая модель представления знаний, так как в ней

имеются средства реализации всех характерных для знаний свойств: внутренней интерпретации, структурированности, семантической метрики и активности.

Характерной особенностью семантических сетей является обязательное наличие трех типов отношений:

класс – элемент класса (цветок – роза);

свойство – значение (цвет – желтый);

пример элемента класса (роза – чайная).

Можно предложить несколько классификаций семантических сетей, связанных с типами отношений между понятиями.

По количеству типов отношений:

однородные (с единственным типом отношений);

неоднородные (с различными типами отношений).

По типам отношений:

бинарные (в которых отношения связывают два объекта);

N-арные (в которых есть специальные отношения связывающие более двух понятий).

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»…);

количественные (больше, меньше, равно…);

пространственные (далеко от, близко от, за, под, над …);

временные (раньше, позже, в течение…);

атрибутивные связи (иметь свойство, иметь значение);

логические (И, ИЛИ, НЕ);

лингвистические связи и д.р.

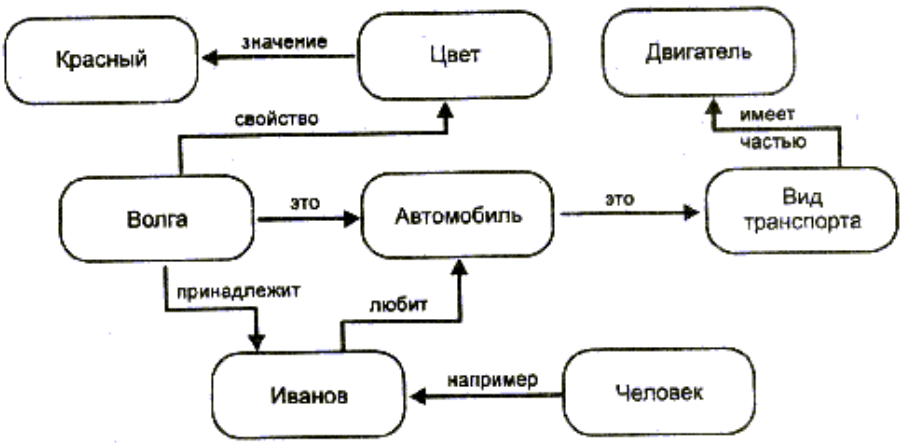


Рис. Семантическая сеть

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

Данная модель представления знаний была предложена американским психологом Куиллианом. Основным ее преимуществом является то, что она более других соответствует современным представлениям об организации долговременной памяти человека.

Недостатком этой модели является сложность организации процедуры поиска вывода на семантической сети.

Достоинства сетевых моделей: большие выразительные возможности; наглядность системы знаний, представленной графически; близость структуры сети, представляющей систему знаний, семантической структуре фраз на естественном языке; соответствие современным представлениям об организации долговременной памяти человека.

Недостатки: сетевая модель не дает (точнее, не содержит) ясного представления о структуре предметной области, которая ей соответствует, поэтому формирование и модификация такой модели затруднительны; сетевые модели представляют собой пассивные структуры, для обработки которых необходим специальный аппарат формального вывода и планирования. Проблема поиска решения в базе знаний типа семантической сети сводится к задаче поиска фрагмента сети, соответствующего некоторой подсети поставленной задачи. Это, в

свою очередь, обусловливает еще один недостаток модели — сложность поиска вывода на семантических сетях.

2.4 Фреймовые модели

Термин *фрейм* (frame — каркас, рамка) предложен М. Минским в 70-е годы для обозначения структуры знаний для восприятия пространственных сцен. Эта модель, как и семантическая сеть, имеет глубокое психологическое обоснование. Под фреймом понимается абстрактный образ или ситуация. В психологии и философии известно понятие абстрактного образа. Например, слово «комната» вызывает образ комнаты — «жилое помещение с четырьмя стенами, полом, потолком, окнами и дверью». Из этого описания ничего нельзя убрать, например, убрав окна, мы получим уже чулан, а не комнату. Но в нем есть «слоты», или «щели», — незаполненные значения некоторых атрибутов — количество окон, цвет стен, высота потолка, покрытие пола и др. Такой образ и называется фреймом (фреймом минимального описания). Фреймом называется также и формализованная модель этого образа.

Фреймовая модель, основанная на теории М. Минского, представляет собой систематизированную в виде единой теории технологическую модель памяти человека и его сознания. Важным элементом в этой теории является понятие фрейма -структуры данных для представления некоторого концептуального объекта. Информация, относящаяся к этому фрейму, содержится в составляющих фрейма — слотах. В отличие от моделей других типов, во фреймовых моделях фиксируется жесткая структура, которая называется протофреймом (фреймом-прототипом, или образцом). В общем случае фрейм определяется следующим образом:

где f имя фрейма; v_i. значение слота, или

(ИМЯ ФРЕЙМА:

(имя 1-го слота: значение 1-го слота),

(имя 2-го слота: значение 2-го слота),

(имя n-го слота: значение n-го слота)).

Значением слота может быть практически что угодно (числа или математические соотношения, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма). В качестве значения слота может выступать набор слотов более низкого уровня, что позволяет во фреймовых представлениях реализовать «принцип матрешки».

В качестве значения слота может выступать имя другого фрейма; так образуются сети фреймов. Все фреймы взаимосвязаны и образуют единую фреймовую структуру, в которой органически объединены декларативные и процедурные знания. Это дает возможность достаточно естественно производить композицию и декомпозицию информационных структур аналогично тому, как это делал бы человек при описании структуры своих знаний.

Кроме фреймов-образцов, или прототипов, хранящихся в базе знаний, различают *фреймы-экземпляры,* которые создаются для отображения реальных ситуаций на основе поступающих данных. При конкретизации (означивании) фрейма ему и слотам присваиваются конкретные имена, и происходит заполнение слотов. Таким образом, из протофреймов получаются фреймы-экземпляры.

Если в качестве значений слотов использовать реальные данные из таблицы, то получится фрейм-экземпляр.

Важнейшим свойством фреймов является заимствованное из теории семантических сетей наследование свойств. И во фреймах, и в семантических сетях наследование происходит по АКО-связям (от *A Kind Of* = это). Слот АКО указывает на фрейм более высокого уровня иерархии, откуда неявно наследуются, то есть переносятся значения аналогичных слотов, причем наследование свойств может быть частичным.

Фреймовые модели является достаточно универсальными, поскольку позволяют отобразить все многообразие знаний о мире:

через фреймы-структуры для обозначений 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 т. д.).

Основными достоинствами модели фреймов как модели представления знаний являются способность отражать концептуальную основу организации памяти человека, а также естественность, наглядность представления, модульность, поддержка возможности использования значений слотов по умолчанию. Однако фрейм-представление является не конкретным языком представления знаний, а некоторой идеологической концепцией, реализуемой по-разному в различных языках.

Основным недостатком фреймовых моделей является отсутствие механизмов управления выводом. Отчасти этот недостаток устраняется при помощи присоединенных процедур, реализуемых силами пользователя системы. Рассмотренные модели представления знаний во многом близки между собой. По сути, они обладают одинаковыми возможностями описывать и представлять знания. Разница состоит лишь в том, насколько удобно и естественно представлять те или иные знания в виде логических формул, семантических сетей, фреймов или продукций.

Обобщая анализ моделей представления знаний, можно сделать два основных вывода:

Невозможно дать универсальные рекомендации по выбору модели. Выбор конкретной модели определяется возможностью и удобством представления исследуемой проблемной области с учетом необходимости не только представления, но и использования знаний. Однако чаще используются эвристические, а не логические модели представления знаний.

Наиболее мощными оказываются смешанные представления.

2.5. Вывод на знаниях

Несмотря на все недостатки, наибольшее распространение получила продукционная модель представления знаний. При использовании

продукционной модели база знаний состоит из набора правил. *Программа, управляющая перебором правил, называется машиной вывода.*

Машина вывода (интерпретатор правил) выполняет две функции: во-первых, Просмотр существующих фактов из рабочей памяти (базы данных) и правил из базы знаний и добавление (по мере возможности) в рабочую память новых фактов и, во-вторых, определение порядка просмотра и применения правил. Этот механизм управляет процессом консультации, сохраняя для пользователя информацию о полученных заключениях, и запрашивает у него информацию, когда для срабатывания очередного правила в рабочей памяти оказывается недостаточно данных.

В подавляющем большинстве систем, основанных на знаниях, механизм вывода представляет собой небольшую по объему программу и включает два компонента — один реализует собственно вывод, другой управляет этим процессом. Действие *компонента вывода* основано на применении правила, называемого *modus ponens.*

Правило modus ponens. Если известно, что истинно утверждение A и существует правило вида «если A, то B», тогда утверждение B также истинно.

Правила срабатывают, когда находятся факты, удовлетворяющие их левой части: если истинна посылка, то должно быть истинно и заключение.

Компонент вывода должен функционировать даже при недостатке информации. Полученное решение может и не быть точным, однако система не должна останавливаться из-за того, что отсутствует какая-либо часть входной информации

Управляющий компонент определяет порядок применения правил и выполняет четыре функции.

1.Сопоставление

2.Выбор

3.Срабатывание

4.Действие

Интерпретатор продукций работает циклически. В каждом цикле он просматривает все правила, чтобы выявить те, посылки которых совпадают с известными на данный момент фактами из рабочей памяти. После выбора правило срабатывает, его заключение заносится в рабочую память, и затем цикл повторяется сначала.

В одном цикле может сработать только одно правило. Если несколько правил успешно сопоставлены с фактами, то интерпретатор производит выбор по определенному критерию единственного правила, которое срабатывает в данном цикле. Цикл работы интерпретатора схематически представлен на рисунке 2.1.

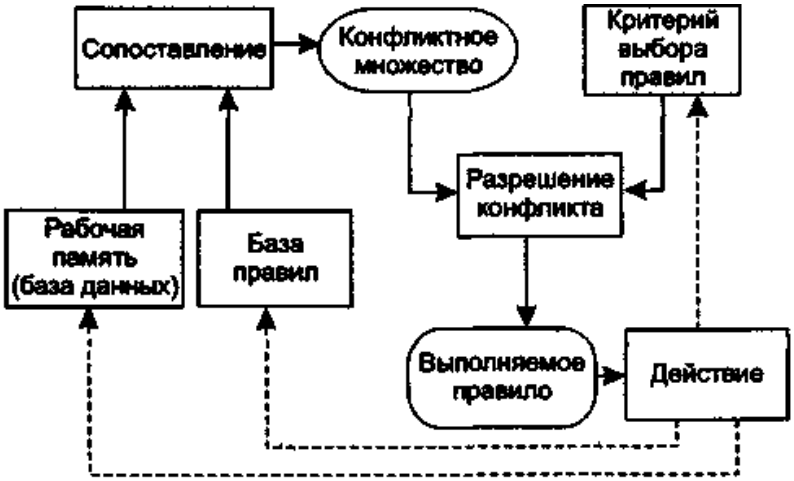


Рисунок 2.1 Цикл работы интерпретатора

От выбранного метода поиска, то есть стратегии вывода, будет зависеть порядок срабатывания правил. Процедура выбора сводится к определению Поиска и способа его осуществления. Процедуры, реализующие поиск, обычно «зашиты» в механизм вывода, поэтому в большинстве систем инженеры знаний не имеют к ним доступа и, следовательно, не могут в них ничего изменять по своему желанию.

При разработке стратегии управления выводом важно определить два вопроса:

Какую точку в пространстве состояний принять в качестве исходной? От выбора этой точки зависит и метод осуществления поиска в прямом или обратном направлении.

Какими методами можно повысить эффективность поиска решения? Эти методы определяются выбранной стратегией перебора глубину, в ширину, по подзадачам или иначе.

Тема 3: Теория и техника приобретения знаний

1. Теоретические аспекты извлечения знаний
2. Эволюция систем приобретения знаний
3. Современное состояние автоматизированных систем приобретения знаний

3.1Теоретические аспекты извлечения знаний

Приобретением знаний называется выявление знаний из источников и преобразование их в нужную форму, а также перенос в базу знаний ИнС. Источниками знаний могут быть:

а) книги, архивные документы, содержимое других баз знаний и т.п., т.е. некоторые объективизированные знания, приведенные к форме, которая делает их доступными для потребителя;

б) экспертные знания, которые имеются у специалистов, но не зафиксированы во внешних по отношению к ним хранилищах (экспертные знания являются субъективными)',

в) эмпирические знания (также субъективный вид знаний), которые получаются путем наблюдения за окружающей средой (если у ИнС есть средства наблюдения)(рис.3.1).

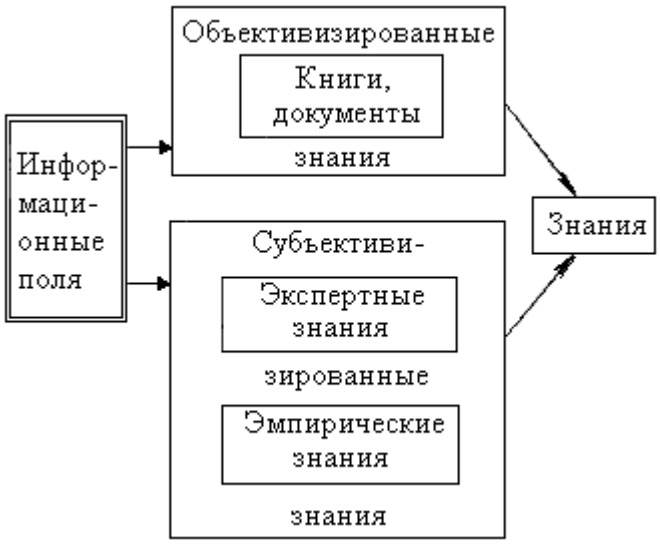


Рисунок 3.1 – Схема приобретения знаний

Ввод в БЗ объективизированных знаний не представляет особой проблемы, выявление и ввод субъективных и особенно экспертных знаний достаточно трудны. Чтобы разработать методологию приобретения субъективных знаний, получаемых от эксперта, надо четко различать две формы репрезентации (представления) знаний.

Одна форма связана с тем, как и в каких моделях хранятся эти знания у человека – эксперта. При этом эксперт не всегда осознает полностью, как репрезентированы у него знания. Другая форма связана с тем, как инженер по знаниям (когнитолог), проектирующий ИС, собирается их описывать и представлять. От степени согласованности этих двух форм репрезентации между собой зависит эффективность инженера по знаниям. В когнитивной психологии изучаются формы репрезентации знаний – когнитивные структуры знаний, характерные для человека. Примерами могут служить:

- представление класса понятий через его элементы (например, понятие «птица» репрезентируется рядом «чайка, воробей, скворец,...»)

птица = <чайка, воробей, скворец,...>;

- представление понятий класса с помощью базового прототипа, отражающего наиболее типичные свойства объектов класса (например, понятие «птица» репрезентируется прототипом «нечто с крыльями, клювом, летает,...»)

птица = <нечто с крыльями, с клювом, летает,...>;

- представление с помощью признаков (для понятия «птица», например, наличие крыльев, клюва, двух лап, перьев, ...)

птица = <крылья, клюв, две лапы, перья,...>.

Кроме понятий репрезентируются и отношения между ними. Как правило, отношения между понятиями определяются процедурным способом, а отношения между составляющими понятий (определяющими структуру понятия) – декларативным способом. Наличие двух видов

описаний заставляет в моделях представления знаний одновременно иметь обе компоненты, например семантическую сеть и продукционную систему, как это представлено в когнитивной модели рис. 3.2

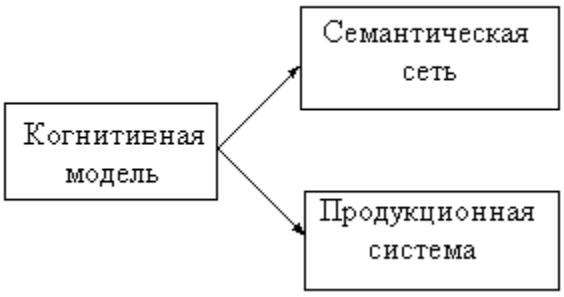


Рисунок 3.2 – Представление когнитивной модели

Проблема приобретения знаний изучается в рамках инженерии знаний. Схема приобретения знаний может быть представлена следующим образом:

Носитель информации → Посредник → Модель знания

Приобретением знаний называют процесс получения знаний от эксперта или каких-либо других источников и передача их в ИнС. Однако наряду с термином «приобретение» сегодня широко используются и другие термины для обозначения этого процесса, например: «извлечение», «получение», «добыча», «формирование знаний». В англоязычной литературе по ИнС используются в основном два термина: acquisition (приобретение) и elicitation (извлечение, выявление, установление).

Для преодоления терминологических различий и достижения общности описания этих сложных процессов воспользуемся предположениями о трех стратегиях получения знаний при разработке ИнС, суть которых представлена на рис. 3.3.

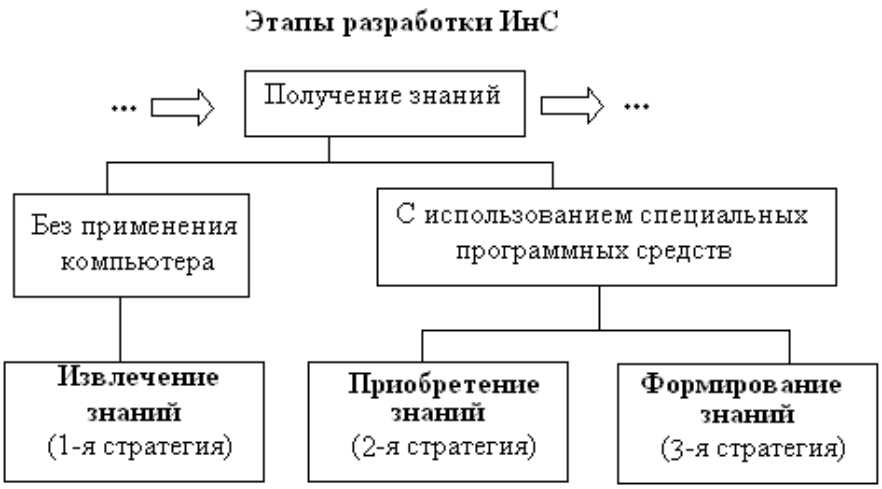


Рисунок 3.3 – Три стратеги получения знаний при разработке ИнС

3.2 Эволюция систем приобретения знаний

Первое поколение таких систем появилось в середине 80-х – это так называемые системы приобретения знаний (СПЗ) (TEIRESIAS [Davis, 1982], SIMER+MIR [Осипов, 1988], АРИАДНА [Моргоев, 1988]). Это средства наполнения так называемых «пустых» ЭС, то есть систем, из БЗ которых изъяты знания (например, EMYCIN – EMPTY MYCIN, опустошенная медицинская ЭС MYCIN со специальной диалоговой системой заполнения базы знаний TEIRESIAS). Их авторы считали, что прямой диалог эксперта с компьютером через СПЗ поможет сократить жизненный цикл разработки. Однако опыт создания и внедрения СПЗ продемонстрировал несовершенство такого подхода.

Основные недостатки СПЗ I поколения:

- Слабая проработка методов извлечения и структурирования знаний.
- Жесткость модели представления знаний, встроенной в СПЗ и связанной с привязкой к программной реализации.
- Ограничения на предметную область.

Таким образом, традиционная схема разработки СПЗ I поколения:

создание конкретной ЭС → опустошение БЗ → разработка СПЗ для новых наполнений БЗ → формирование новой БЗ для другой ЭС

оказалась несостоятельной для промышленного применения.

Второе поколение СПЗ появилось в конце 80-х и было ориентировано на более широкий модельный подход [Gaines, 1989; Борисов, Федоров, Архипов, 1991] с акцентом на предварительном детальном анализе предметной области. Так, в Европе широкое применение получила методология KADS (Knowledge Acquisition and Documentation Structuring) [Wielinga et al., 1989], в основе которой лежит понятие интерпретационной модели, позволяющей процессы извлечения, структурирования и формализации знаний рассматривать как «интерпретацию» лингвистических знаний в другие представления и структуры.

3.3 Современное состояние автоматизированных систем приобретения знаний

Анализ современного состояния программных средств приобретения знаний и поддержки деятельности инженера по знаниям позволяет выявить две группы проблем, характерных для существующих СПЗ:

- методологические проблемы;
- технологические проблемы.

А. Методологические проблемы

Основная проблема, встающая перед разработчиками, – отсутствие теоретического базиса процесса извлечения и структурирования знаний – порождает дочерние более узкие вопросы и казусы на всех этапах создания интеллектуальных систем. Даже тщательно проработанная методология KADS, страдает громоздкостью и явной избыточностью. Ниже перечислены наиболее общие из возникающих проблем в последовательности, соответствующей стадиям жизненного цикла:

- размытость критериев выбора подходящей задачи;
- слабая проработанность теоретических аспектов процессов извлечения знаний (философские, лингвистические, психологические, педагогические, дидактические и другие аспекты), а также отсутствие обоснованной классификации методов извлечения знаний и разброс терминологии;

- отсутствие единого теоретического базиса процедуры структурирования знаний;
- жесткость моделей представления знаний, заставляющая разработчиков обеднять и урезать реальные знания экспертов;
- несовершенство математического базиса моделей представления знаний (дескриптивный, а не конструктивный характер большинства имеющихся математических моделей);
- эмпиричность процедуры выбора программного инструментария и процесса тестирования (отсутствие критериев, разрозненные классификации, etc.).

Б. Технологические проблемы

Большая часть технологических проблем является естественным следствием методологических и порождена ими. Наиболее серьезными из технологических проблем являются:

- отсутствие концептуальной целостности и согласованности между отдельными приемами и методами инженерии знаний;
- недостаток или отсутствие квалифицированных специалистов в области инженерии знаний;
- отсутствие технико-экономических показателей оценки эффективности ЭС (в России);
- несмотря на обилие методов извлечения знаний (фактически более 200), практическая недоступность методических материалов по практике проведения сеансов извлечения знаний;
- явная неполнота и недостаточность имеющихся методов структурирования знаний, отсутствие классификаций и рекомендаций по выбору подходящего метода;
- несмотря на обилие рынка программных средств, недостаток промышленных систем поддержки разработки и их узкая направленность (зависимость от платформы, языка реализации, ограничений предметной области), разрыв между ЯПЗ и языками,

встроенными в «оболочки» ЭС;

- жесткость программных средств, их низкая адаптивность, отсутствие индивидуальной настройки на пользователя и предметную область;
- слабые графические возможности программных средств, недостаточный учет когнитивных и эргономических факторов;
- сложность внедрения ЭС, обусловленная психологическими проблемами персонала и неприятия новой технологии решения задач.

Тема 4: Особенности разработки и использования экспертных систем.

1. Назначение экспертных систем.
2. Структура ЭС
3. Этапы разработки ЭС

4.1 Назначение экспертных систем

В начале восьмидесятых годов в исследованиях по искусственному интеллекту сформировалось самостоятельное направление, получившее название "экспертные системы" (ЭС). Цель исследований по ЭС состоит в разработке программ, которые при решении задач, трудных для эксперта-человека, получают результаты, не уступающие по качеству и эффективности решениям, получаемым экспертом. Исследователи в области ЭС для названия своей дисциплины часто используют также термин «инженерия знаний», введенный Е.Фейгенбаумом как "привнесение принципов и инструментария исследований из области искусственного интеллекта в решение трудных прикладных проблем, требующих знаний экспертов".

Программные средства (ПС), базирующиеся на технологии экспертных систем, или инженерии знаний (в дальнейшем будем использовать их как синонимы), получили значительное распространение в мире. Важность экспертных систем состоит в следующем:

- технология экспертных систем существенно расширяет круг практически значимых задач, решаемых на компьютерах, решение которых приносит значительный экономический эффект;
- технология ЭС является важнейшим средством в решении глобальных проблем традиционного программирования: длительность и, следовательно, высокая стоимость разработки сложных приложений;
- высокая стоимость сопровождения сложных систем, которая часто в несколько раз превосходит стоимость их разработки; низкий уровень повторной используемости программ и т.п.;
- объединение технологии ЭС с технологией традиционного программирования добавляет новые качества к программным продуктам за счет: обеспечения динамичной модификации приложений пользователем, а не программистом; большей "прозрачности" приложения (например, знания хранятся на ограниченном ЕЯ, что не требует комментариев к знаниям, упрощает обучение и сопровождение); лучшей графики; интерфейса и взаимодействия.

По мнению ведущих специалистов, в недалекой перспективе ЭС найдут следующее применение:

- ЭС будут играть ведущую роль во всех фазах проектирования, разработки, производства, распределения, продажи, поддержки и оказания услуг;
- технология ЭС, получившая коммерческое распространение, обеспечит революционный прорыв в интеграции приложений из готовых интеллектуально-взаимодействующих модулей.

ЭС предназначены для так называемых неформализованных задач, т.е. ЭС не отвергают и не заменяют традиционного подхода к разработке программ, ориентированного на решение формализованных задач.

Неформализованные задачи обычно обладают следующими

особенностями:

- ошибочностью, неоднозначностью, неполнотой и противоречивостью исходных данных;
- ошибочностью, неоднозначностью, неполнотой и противоречивостью знаний о проблемной области и решаемой задаче;
- большой размерностью пространства решения, т.е. перебор при поиске решения весьма велик;
- динамически изменяющимися данными и знаниями.

Следует подчеркнуть, что неформализованные задачи представляют большой и очень важный класс задач. Многие специалисты считают, что эти задачи являются наиболее массовым классом задач, решаемых ЭВМ.

Экспертные системы и системы искусственного интеллекта отличаются от систем обработки данных тем, что в них в основном используются символьный (а не числовой) способ представления, символьный вывод и эвристический поиск решения (а не исполнение известного алгоритма).

Экспертные системы применяются для решения только трудных практических (не игрушечных) задач. По качеству и эффективности решения экспертные системы не уступают решениям эксперта-человека. Решения экспертных систем обладают *"прозрачностью",* т.е. могут быть объяснены пользователю на качественном уровне. Это качество экспертных систем обеспечивается их способностью рассуждать о своих знаниях и умозаключениях. Экспертные системы способны пополнять свои знания в ходе взаимодействия с экспертом. Необходимо отметить, что в настоящее время технология экспертных систем используется для решения различных типов задач (интерпретация, предсказание, диагностика, планирование, конструирование, контроль, отладка, инструктаж, управление) в самых разнообразных проблемных областях, таких, как финансы, нефтяная и газовая промышленность, энергетика, транспорт, фармацевтическое производство, космос, металлургия, горное дело, химия, образование,

целлюлозно-бумажная промышленность, телекоммуникации и связь и др.

Коммерческие успехи к фирмам-разработчикам систем искусственного интеллекта (СИИ) пришли не сразу. На протяжении 1960–1985 гг. успехи ИИ касались в основном исследовательских разработок, которые демонстрировали пригодность СИИ для практического использования. Начиная примерно с 1985 г. (в массовом масштабе с 1988–1990 гг.), в первую очередь ЭС, а в последние годы системы, воспринимающие естественный язык (ЕЯ-системы), и нейронные сети (НС) стали активно использоваться в коммерческих приложениях.

Следует обратить внимание на то, что некоторые специалисты (как правило, специалисты в программировании, а не в ИИ) продолжают утверждать, что ЭС и СИИ не оправдали возлагавшихся на них ожиданий и умерли. Причины таких заблуждений состоят в том, что эти авторы рассматривали ЭС как альтернативу традиционному программированию, т.е. они исходили из того, что ЭС в одиночестве (в изоляции от других программных средств) полностью решают задачи, стоящие перед заказчиком. Надо отметить, что на заре появления ЭС специфика используемых в них языков, технологии разработки приложений и используемого оборудования (например, Lisp-машины) давала основания предполагать, что интеграция ЭС с традиционными, программными системами является сложной и, возможно, невыполнимой задачей при ограничениях, накладываемых реальными приложениями. Однако в настоящее время коммерческие инструментальные средства (ИС) для создания ЭС разрабатываются в полном соответствии с современными технологическими тенденциями традиционного программирования, что снимает проблемы, возникающие при создании интегрированных приложений.

Причины, приведшие СИИ к коммерческому успеху, следующие.

Интегрированность. Разработаны инструментальные средства искусственного интеллекта (ИС ИИ), легко интегрирующиеся с другими информационными технологиями и средствами (с CASE, СУБД,

контроллерами, концентраторами данных и т.п.).

Открытость и переносимость. ИС ИИ разрабатываются с соблюдением стандартов, обеспечивающих открытость и переносимость.

Использование языков традиционного программирования и рабочих станций. Переход от ИС ИИ, реализованных на языках ИИ (Lisp, Prolog и т.п.), к ИС ИИ, реализованным на языках традиционного программирования (C, C++ и т.п.), упростил обеспечение интегрированности, снизил требования приложений ИИ к быстродействию ЭВМ и объемам оперативной памяти. Использование рабочих станций (вместо ПК) резко увеличило круг приложений, которые могут быть выполнены на ЭВМ с использованием ИС ИИ.

Архитектура клиент-сервер. Разработаны ИС ИИ, поддерживающие распределенные вычисления по архитектуре клиент-сервер, что позволило: снизить стоимость оборудования, используемого в приложениях, децентрализовать приложения, повысить надежность и общую производительность (так как сокращается количество информации, пересылаемой между ЭВМ, и каждый модуль приложения выполняется на адекватном ему оборудовании).

Проблемно/предметно-ориентированные ИС ИИ. Переход от разработок ИС ИИ общего назначения (хотя они не утратили свое значение как средство для создания ориентированных ИС) к проблемно/предметно-ориентированным ИС ИИ [9] обеспечивает: сокращение сроков разработки приложений; увеличение эффективности использования ИС; упрощение и ускорение работы эксперта; повторную используемость информационного и программного обеспечения (объекты, классы, правила, процедуры).

4.2 Структура экспертных систем

Типичная статическая ЭС состоит из следующих основных компонентов:

- решателя (интерпретатора);
- рабочей памяти (РП), называемой также базой данных (БД);

- базы знаний (БЗ);
- компонентов приобретения знаний;
- объяснительного компонента;
- диалогового компонента.

База данных (рабочая память) предназначена для хранения исходных и промежуточных данных решаемой в текущий момент задачи. Этот термин совпадает по названию, но не по смыслу с термином, используемым в информационно-поисковых системах (ИПС) и системах управления базами данных (СУБД) для обозначения всех данных (в первую очередь долгосрочных), хранимых в системе.

База знаний (БЗ) в ЭС предназначена для хранения долгосрочных данных, описывающих рассматриваемую область (а не текущих данных), и правил, описывающих целесообразные преобразования данных этой области.

Решатель, используя исходные данные из рабочей памяти и знания из БЗ, формирует такую последовательность правил, которые, будучи примененными к исходным данным, приводят к решению задачи.

Компонент приобретения знаний автоматизирует процесс наполнения ЭС знаниями, осуществляемый пользователем-экспертом.

Объяснительный компонент объясняет, как система получила решение задачи (или почему она не получила решение) и какие знания она при этом использовала, что облегчает эксперту тестирование системы и повышает доверие пользователя к полученному результату.

Диалоговый компонент ориентирован на организацию дружественного общения с пользователем как в ходе решения задач, так и в процессе приобретения знаний и объяснения результатов работы.

В разработке ЭС участвуют представители следующих специальностей:

эксперт в проблемной области, задачи которой будет решать ЭС;

инженер по знаниям - специалист по разработке ЭС (используемые им технологию, методы называют технологией (методами) инженерии знаний);

программист по разработке инструментальных средств (ИС), предназначенных для ускорения разработки ЭС.

Необходимо отметить, что отсутствие среди участников разработки инженеров по знаниям (т. е. их замена программистами) либо приводит к неудаче процесс создания ЭС, либо значительно удлиняет его.

Эксперт определяет знания (данные и правила), характеризующие проблемную область, обеспечивает полноту и правильность введенных в ЭС знаний.

Инженер по знаниям помогает эксперту выявить и структурировать знания, необходимые для работы ЭС; осуществляет выбор того ИС, которое наиболее подходит для данной проблемной области, и определяет способ представления знаний в этом ИС; выделяет и программирует (традиционными средствами) стандартные функции (типичные для данной проблемной области), которые будут использоваться в правилах, вводимых экспертом.

Программист разрабатывает ИС (если ИС разрабатывается заново), содержащее в пределе все основные компоненты ЭС, и осуществляет его сопряжение с той средой, в которой оно будет использовано.

Экспертная система работает в двух режимах: режиме приобретения знаний и в режиме решения задачи (называемом также режимом консультации или режимом использования ЭС).

В режиме приобретения знаний общение с ЭС осуществляет (через посредничество инженера по знаниям) эксперт. В этом режиме эксперт, используя компонент приобретения знаний, наполняет систему знаниями, которые позволяют ЭС в режиме решения самостоятельно (без эксперта) решать задачи из проблемной области. Эксперт описывает проблемную область в виде совокупности данных и правил. Данные определяют объекты, их характеристики и значения, существующие в области экспертизы. Правила определяют способы манипулирования с данными, характерные для рассматриваемой области.

Отметим, что режиму приобретения знаний в традиционном подходе к разработке программ соответствуют этапы алгоритмизации, программирования и отладки, выполняемые программистом. Таким образом, в отличие от традиционного подхода в случае ЭС разработку программ осуществляет не программист, а эксперт (с помощью ЭС), не владеющий программированием.

В режиме консультации общение с ЭС осуществляет конечный пользователь, которого интересует результат и (или) способ его получения. Необходимо отметить, что в зависимости от назначения ЭС пользователь может не быть специалистом в данной проблемной области (в этом случае он обращается к ЭС за результатом, не умея получить его сам), или быть специалистом (в этом случае пользователь может сам получить результат, но он обращается к ЭС с целью либо ускорить процесс получения результата, либо возложить на ЭС рутинную работу). В режиме консультации данные о задаче пользователя после обработки их диалоговым компонентом поступают в рабочую память. Решатель на основе входных данных из рабочей памяти, общих данных о проблемной области и правил из БЗ формирует решение задачи. ЭС при решении задачи не только исполняет предписанную последовательность операции, но и предварительно формирует ее. Если реакция системы не понятна пользователю, то он может потребовать объяснения: «Почему система задает тот или иной вопрос?», «как ответ, собираемый системой, получен?».

4.3 Этапы разработки экспертных систем

Разработка ЭС имеет существенные отличия от разработки обычного программного продукта. Опыт создания ЭС показал, что использование при их разработке методологии, принятой в традиционном программировании, либо чрезмерно затягивает процесс создания ЭС, либо вообще приводит к отрицательному результату.

Использовать ЭС следует только тогда, когда разработка ЭС *возможна, оправдана и* методы инженерии знаний *соответствуют*

решаемой задаче. Чтобы разработка ЭС была *возможной* для данного приложения, необходимо одновременное выполнение по крайней мере следующих требований:

- существуют эксперты в данной области, которые решают задачу значительно лучше, чем начинающие специалисты;
- эксперты сходятся в оценке предлагаемого решения, иначе нельзя будет оценить качество разработанной ЭС;
- эксперты способны вербализовать (выразить на естественном языке) и объяснить используемые ими методы, в противном случае трудно рассчитывать на то, что знания экспертов будут "извлечены" и вложены в ЭС;
- решение задачи требует только рассуждений, а не действий;
- задача не должна быть слишком трудной (т.е. ее решение должно занимать у эксперта несколько часов или дней, а не недель);
- задача хотя и не должна быть выражена в формальном виде, но все же должна относиться к достаточно "понятной" и структурированной области, т.е. должны быть выделены основные понятия, отношения и известные (хотя бы эксперту) способы получения решения задачи;
- решение задачи не должно в значительной степени использовать "здравый смысл" (т.е. широкий спектр общих сведений о мире и о способе его функционирования, которые знает и умеет использовать любой нормальный человек), так как подобные знания пока не удается (в достаточном количестве) вложить в системы искусственного интеллекта.

Использование ЭС в данном приложении может быть возможно, но не оправдано. Применение ЭС может быть *оправдано* одним из следующих факторов:

- решение задачи принесет значительный эффект, например экономический;

- использование человека-эксперта невозможно либо из-за недостаточного количества экспертов, либо из-за необходимости выполнять экспертизу одновременно в различных местах;
- использование ЭС целесообразно в тех случаях, когда при передаче информации эксперту происходит недопустимая потеря времени или информации;
- использование ЭС целесообразно при необходимости решать задачу в окружении, враждебном для человека.

Приложение *соответствует* методам ЭС, если решаемая задача обладает совокупностью следующих характеристик:

- задача может быть естественным образом решена посредством манипуляции с символами (т.е. с помощью символических рассуждений), а не манипуляций с числами, как принято в математических методах и в традиционном программировании;
- задача должна иметь эвристическую, а не алгоритмическую природу, т.е. ее решение должно требовать применения эвристических правил. Задачи, которые могут быть гарантированно решены (с соблюдением заданных ограничений) с помощью некоторых формальных процедур, не подходят для применения ЭС;
- задача должна быть достаточно сложна, чтобы оправдать затраты на разработку ЭС. Однако она не должна быть чрезмерно сложной (решение занимает у эксперта часы, а не недели), чтобы ЭС могла ее решать;
- задача должна быть достаточно узкой, чтобы решаться методами ЭС, и практически значимой.

При разработке ЭС, как правило, используется концепция "быстрого прототипа". Суть этой концепции состоит в том, что разработчики не пытаются сразу построить конечный продукт. На начальном этапе они создают прототип (прототипы) ЭС. Прототипы должны удовлетворять двум противоречивым требованиям: с одной стороны, они должны решать

типичные задачи конкретного приложения, а с другой - время и трудоемкость их разработки должны быть весьма незначительны, чтобы можно было максимально запараллелить процесс накопления и отладки знаний (осуществляемый экспертом) с процессом выбора (разработки) программных средств (осуществляемым инженером по знаниям и программистом). Для удовлетворения указанным требованиям, как правило, при создании прототипа используются разнообразные средства, ускоряющие процесс проектирования.

Прототип должен продемонстрировать пригодность методов инженерии знаний для данного приложения. В случае успеха эксперт с помощью инженера по знаниям расширяет знания прототипа о проблемной области. При неудаче может потребоваться разработка нового прототипа или разработчики могут прийти к выводу о непригодности методов ЭС для данного приложения. По мере увеличения знаний прототип может достигнуть такого состояния, когда он успешно решает все задачи данного приложения. Преобразование прототипа ЭС в конечный продукт обычно приводит к перепрограммированию ЭС на языках низкого уровня, обеспечивающих как увеличение быстродействия ЭС, так и уменьшение требуемой памяти. Трудоемкость и время создания ЭС в значительной степени зависят от типа используемого инструментария.

В ходе работ по созданию ЭС сложилась определенная технология их разработки, включающая шесть следующих этапов: идентификацию, концептуализацию, формализацию, выполнение, тестирование, опытную эксплуатацию.

На этапе *идентификации* определяются задачи, которые подлежат решению, выявляются цели разработки, определяются эксперты и типы пользователей.

На этапе *концептуализации* проводится содержательный анализ проблемной области, выявляются используемые понятия и их взаимосвязи, определяются методы решения задач.

На этапе *формализации* выбираются ИС и определяются способы представления всех видов знаний, формализуются основные понятия, определяются способы интерпретации знаний, моделируется работа системы, оценивается адекватность целям системы зафиксированных понятий, методов решений, средств представления и манипулирования знаниями.

На этапе *выполнения* осуществляется наполнение экспертом базы знаний. В связи с тем, что основой ЭС являются знания, данный этап является наиболее важным и наиболее трудоемким этапом разработки ЭС. Процесс приобретения знаний разделяют на извлечение знаний из эксперта, организацию знаний, обеспечивающую эффективную работу системы, и представление знаний в виде, понятном ЭС. Процесс приобретения знаний осуществляется инженером по знаниям на основе анализа деятельности эксперта по решению реальных задач.

Тема 5: Нечеткие знания и способы их обработки

1. Основы теории нечетких множеств
2. Виды нечеткости знаний.
3. Операции с нечеткими знаниями.

5.1Основы теории нечетких множеств

При попытке формализовать человеческие знания исследователи вскоре столкнулись с проблемой, затруднявшей использование традиционного математического аппарата для их описания. 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 (например, «Одним из возможных признаков гриппа является *высокая* температура»).

Кроме того, в задачах, решаемых интеллектуальными системами, часто приходится пользоваться неточными знаниями, которые не могут быть интерпретированы как полностью истинные или ложные (логические

true/false или 0/1). Существуют знания, достоверность которых выражается некоторой промежуточной цифрой, например 0.7.

Как, не разрушая свойства размытости и неточности, представлять подобные знания формально? Для разрешения таких проблем в начале 70-х американский математик *Лотфи Заде* предложил формальный аппарат нечеткой (fuzzy) алгебры и нечеткой логики. Позднее это направление получило широкое распространение и положило начало одной из ветвей ИИ под названием – *мягкие вычисления* (soft computing).

Л. Заде ввел одно из главных понятий в нечеткой логике – понятие лингвистической переменной.

Лингвистическая переменная (ЛП) – это переменная, значение которой определяется набором вербальных (то есть словесных) характеристик некоторого свойства.

Например, ЛП «рост» определяется через набор *{карликовый, низкий, средний, высокий, очень высокий}*.

Значения лингвистической переменной (ЛП) определяются через так называемые *нечеткие множества* (НМ), которые в свою очередь определены на некотором *базовом* наборе значений или базовой числовой шкале, имеющей размерность. Каждое значение ЛП определяется как нечеткое множество (например, НМ «низкий рост»).

Нечеткое множество определяется через некоторую базовую шкалу B и функцию принадлежности НМ – $\mu(x)$, $x \in B$ принимающую значения на интервале [0...1]. Таким образом, нечеткое множество B — это совокупность пар вида $(x, \mu(x))$, где $x \in B$. Часто встречается и такая запись:

$$B = \sum_{i=1}^{n} \frac{x_i}{\mu(x_i)}$$

где x_i – i –е значение базовой шкалы.

Функция принадлежности определяет субъективную *степень уверенности* эксперта в том, что данное конкретное значение базовой шкалы соответствует определяемому НМ. Эту функцию не стоит путать с

вероятностью, носящей объективный характер и подчиняющейся другим математическим зависимостям

Например, для двух экспертов определение НМ «высокая» для ЛП «цена автомобиля» в условных единицах может существенно отличаться в зависимости от их социального и финансового положения.

«Высокая_цена_автомобиля_1» - {50000/1 + 25000/0 8 + 10000/0 6 + 5000/0.4} «Высокая_цена_автомобиля_2» - {25000/1 + 10000/0.8 + 5000/0 7 + 3000/0.4}

Пусть перед нами стоит задача интерпретации значений ЛП *«возраст»,* таких как – «молодой» возраст, «преклонный» возраст или «переходный» возраст. Определим «возраст» как ЛП (рис. 5.1). Тогда «молодой», «преклонный», «переходный» будут значениями этой лингвистической переменной более полно, базовый набор значений ЛП «возраст» следующий В – {младенческий, детский, юный, молодой, зрелый, преклонный, старческий}

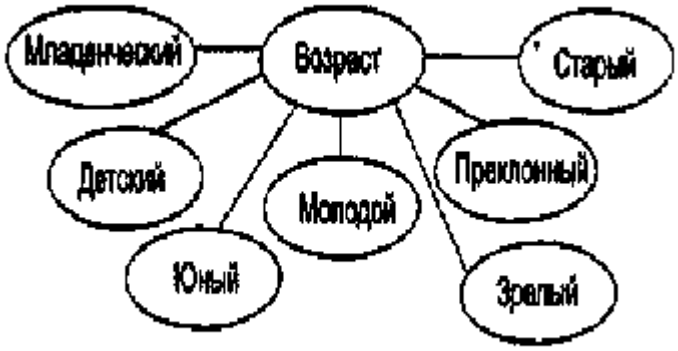


Рисунок 5.1 – Лингвистическая переменная «возраст» и нечеткие множества, определяющие ее значения

Для ЛП «возраст» базовая шкала – это числовая шкала от 0 до 120, обозначающая количество прожитых лет, а функция принадлежности определяет, насколько мы уверены в том, что данное количество лет можно отнести к данной категории возраста на рис. 5.2 отражено, как одни и те же значения базовой шкалы могут участвовать в определении различных НМ.

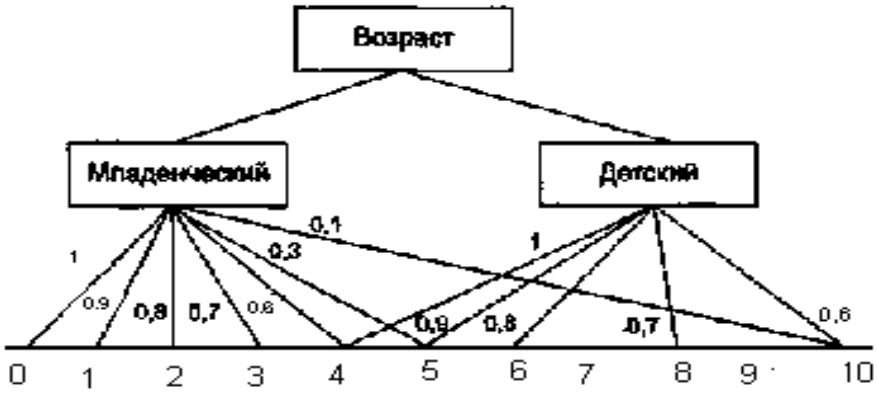


Рисунок 5.2 – Формирование нечетких множеств

Например, определить значение НМ «младенческий возраст» можно так

$$"младенческий" = \left\{ \frac{0,5}{1} + \frac{1}{0,9} + \frac{2}{0,8} + \frac{3}{0,7} + \frac{4}{0,6} + \frac{5}{0,3} + \frac{10}{0,1} \right\}$$

5.2 Виды нечеткости знаний

Некорректные задачи существуют практически в любой проблемной области. Там, где есть некорректные и слабоструктурированные задачи, можно ожидать эффекта от применения интеллектуальных систем. В области некорректных задач точные знания о проблеме получить невозможно или нельзя это сделать сразу, поэтому там применяется подход, суть которого состоит в постепенном приближении к полному набору необходимых знаний. При этом используются методы представления нечетких знаний и механизмы вывода, работающие в их среде.

Смысл термина нечеткость многозначен. Трудно претендовать на исчерпывающее определение этого понятия, поэтому рассмотрим лишь основные его компоненты, к которым относятся следующие:

- недетерминированность выводов;
- многозначность;
- ненадежность;
- неполнота;
- неточность.

5.3 Операции с нечеткими знаниями

Для операций с нечеткими знаниями, выраженными при помощи лингвистических переменных, существует много различных способов. Эти способы являются в основном эвристиками.

Операция «ИЛИ» часто задается так:

$$\mu(x) = \max(\mu_1(x), \mu_2(x))$$

или так: $\mu(x) = \mu_1(x) + \mu_2(x) - \mu_1(x) \cdot \mu_2(x)$

(вероятностный подход).

Усиление или ослабление лингвистических понятий достигается введением специальных квантификаторов. Например, если понятие «старческий возраст» определяется так:

$$\left\{ \frac{60}{0,6} + \frac{70}{0,8} + \frac{80}{0,9} + \frac{90}{1} \right\}$$

то понятие «очень старческий возраст» определяется так

$$con(A) = A^2 = \sum_i \frac{x_i}{\mu_i^2}$$

$$\left\{ \frac{60}{0,36} + \frac{70}{0,64} + \frac{80}{0,81} + \frac{90}{1} \right\}$$

Для вывода на нечетких множествах используются специальные отношения и операции над ними.

Одним из первых применений теории НМ стало использование коэффициентов уверенности для вывода рекомендаций медицинской системы MYCIN. Этот метод использует несколько эвристических приемов. Он стал примером обработки нечетких знаний, повлиявших на последующие системы.

В настоящее время в большинство инструментальных средств разработки систем, основанных на знаниях, включены элементы работы с НМ, кроме того, разработаны специальные программные средства реализации так называемого нечеткого, вывода, например «оболочка» FuzzyCLIPS.

Тема 6: Программный инструментарий разработки систем основанных на знаниях.

1. Языки программирования для ИИ и языки представления знаний.

2. Инструментальные пакеты для ИИ
3. Применение языка ПРОЛОГ для представления знаний

6.1 Языки программирования для ИИ и языки представления знаний

Необходимость использования средств автоматизации программирования прикладных систем, ориентированных на знания, и в частности ЭС, была осознана разработчиками этого класса программного обеспечения ЭВМ уже давно. По существу, средства поддержки разработки интеллектуальных систем в своем развитии прошли основные стадии, характерные для систем автоматизации программирования.

Оценивая данный процесс с сегодняшних позиций, можно указать в этой области две тенденции. Первая из них как бы повторяет «классический» путь развития средств автоматизации программирования: автокоды – языки высокого уровня – языки сверхвысокого уровня – языки спецификаций. Условно эту тенденцию можно назвать восходящей стратегией в области создания средств автоматизации разработки интеллектуальных систем. Вторая тенденция, нисходящая, связывается со специальными средствами, уже изначально ориентированными на определенные классы задач и методов ИИ. В конце концов, обе эти тенденции, взаимно обогатив друг друга, должны привести к созданию мощного и гибкого инструментария интеллектуального программирования. Но для настоящего этапа в этой области характерна концентрация усилий в следующих направлениях:

1. Разработка систем представления знаний (СПЗ) путем прямого использования широко распространенных языков обработки символьной информации и, все чаще, языков программирования общего назначения.

2. Расширение базисных языков ИИ до систем представления знаний за счет специализированных библиотек и пакетов.

3. Создание языков представления знаний (ЯПЗ), специально ориентированных на поддержку определенных формализмов, и реализация соответствующих трансляторов с этих языков.

На начальном этапе развития ИИ языков и систем, ориентированных

специально на создание прикладных систем, основанных на знаниях, не существовало. С одной стороны, в то время еще не оформился сам подход, в котором центральное место отводилось бы изложению теории в форме программ, а с другой – сама область ИИ только зарождалась как научное направление. Немаловажным было и то, что появившиеся к тому времени универсальные языки программирования высокого уровня казались адекватным инструментом для создания любых, в том числе и интеллектуальных, систем. Однако сложность и трудоемкость разработки здесь настолько велики, что практически полезные интеллектуальные системы становятся недоступными для реализации. Учитывая вышесказанное, были разработаны языки и системы обработки символьной информации, которые на несколько десятилетий стали основным инструментом программирования интеллектуальных систем.

До недавнего времени наиболее популярным базовым языком реализации систем ИИ вообще и ЭС, в частности, был ЛИСП. Схема развития средств автоматизации программирования интеллектуальных систем представлена на рис. 6.1.

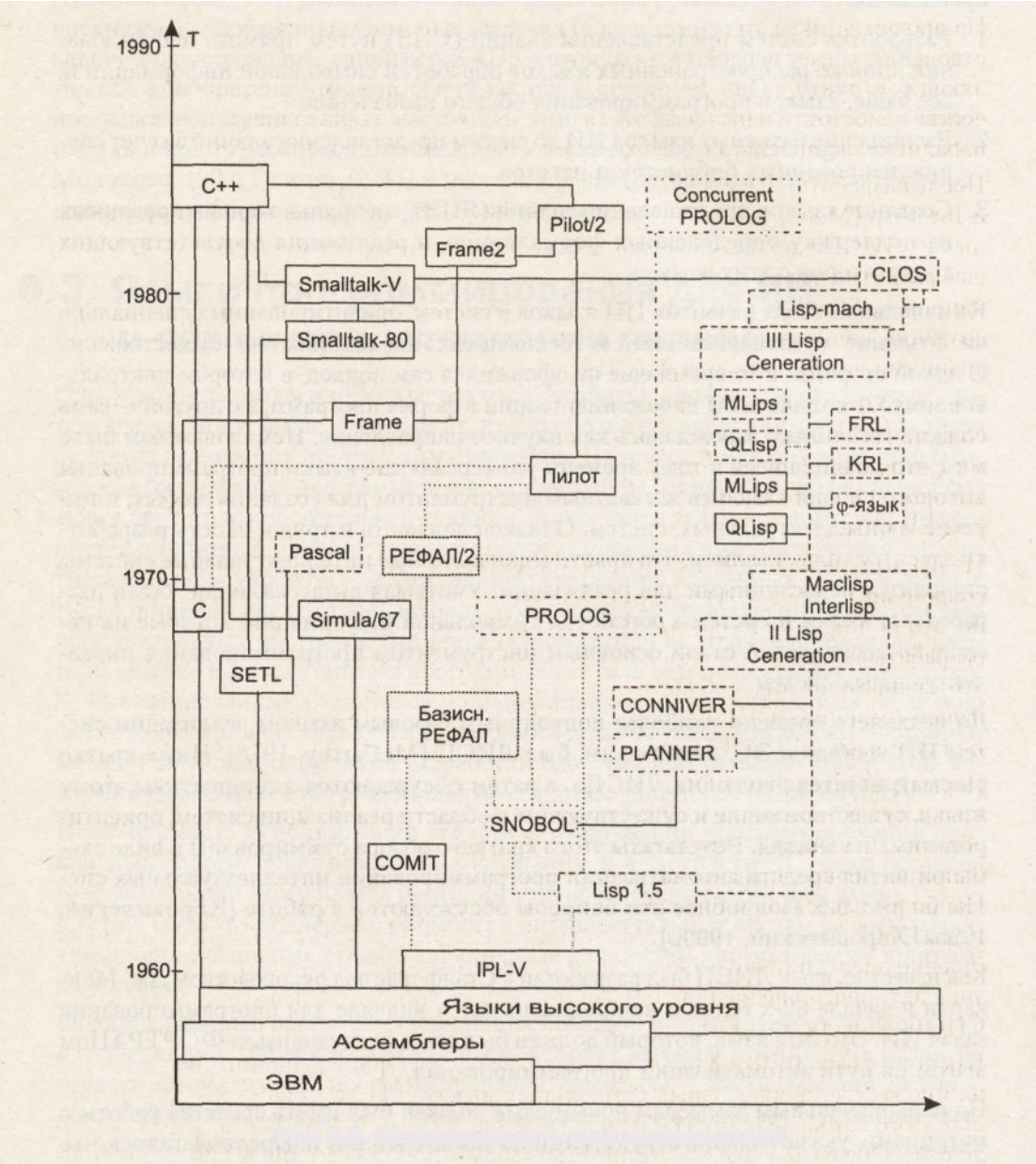


Рисунок 6.1 – Эволюция средств представления знаний

6.2. Инструментальные пакеты для ИИ

Развитые среды автоматизации программирования на базе языков символьной обработки являются необходимым технологическим уровнем систем поддержки разработки прикладных интеллектуальных систем. Как правило, такие среды покрывают (и то частично) подсистему автоматизации проектирования и программирования. Вот почему следующим этапом в развитии инструментальных средств стала ориентация на среды поддержки разработок интеллектуальных систем.

Анализ существующих инструментальных систем показывает, что

сначала в области ИИ более активно велись работы по созданию интеллектуальных систем автоматизированного синтеза исполнительных программ. И это естественно, если иметь в виду, что инструментарий ИИ является, по существу, эволюционным развитием систем автоматизации программирования. При этом основная доля мощности и интеллектуальности такого инструментария связывалась не с его архитектурой, а с функциональными возможностями отдельных компонентов той или иной технологической среды. Большое значение при разработке инструментария для ИИ уделялось и удобству сопряжения отдельных компонентов. Пожалуй, именно здесь были получены впечатляющие результаты и именно здесь наиболее широко использовались последние достижения теории и практики программирования, такие, например, как синтаксически-ориентированное редактирование и инкрементная компиляция. Вместе с тем подавляющее большинство современных инструментальных систем «не знают», что проектирует и реализует с их помощью пользователь. И с этой точки зрения можно сказать, что все такие системы являются не более чем «сундучками» с инструментами, успех использования которых определяется искусством работающего с ними мастера. Примерами подобных сред служит подавляющее большинство инструментальных пакетов и систем-оболочек для создания экспертных систем типа EXSYS, GURU и др. Однако не они определяют на сегодняшний день уровень достижений в этой области. К первому эшелону большинство специалистов относит системы ART, KEE и Knowledge Craft. Заметим, что в последнее время в класс самых мощных и развитых систем вошла и среда G2. Все эти системы, во-первых, отличает то, что это, безусловно, интегрированные среды поддержки разработки интеллектуальных (в первую очередь, экспертных) систем. И вместе с тем для этих систем характерно не эклектичное объединение различных полезных блоков, но тщательно сбалансированный их отбор, что позволило сделать первые шаги от автоматизации программирования систем ИИ к технологическим системам поддержки проектирования сначала экспертных,

а затем и других интеллектуальных систем.

6.3 Применение языка ПРОЛОГ для представления знаний

В отличие от подавляющего большинства других языков Пролог (Prolog) обычно рассматривается в одном контексте с понятием "логическое программирование". По прогнозам, в скором будущем решая задачу, человек не будет задавать последовательность команд на компьютер-ориентированном языке, а будет описывать ее в совершенно абстрактных логических терминах, не оперирующих определениями "байт" или "указатель". То есть он будет создавать модель анализируемой проблемы и пытаться получить положительные или отрицательные результаты этого анализа. Результатом такого анализа может быть, например, оптимизированная программа на процедурном языке (например, C++).

Строго говоря, Пролог не является языком программирования в чистом виде. С одной стороны, это оболочка экспертной системы, с другой - интеллектуальная база данных, что самое важное, не реляционная. Математическая модель, лежащая в основе Пролога, довольно сложна, и по мощности системы формирования запросов к базе с этим языком не сравнится ни одна из коммерческих СУБД.

Фактически Пролог является не процедурным, а декларативным языком. И относится к группе пост объектно-ориентированных языков – функциональным языкам. Человек лишь описывает структуру задачи, а Пролог-машина вывода сама ищет решение. Более того, здесь вообще не существует понятия последовательности команд, все это скрыто в математической модели языка. Однако справедливо заметим, что хотя в идеальной модели Пролог-машины последовательность целей не играет роли, в ее реализациях от порядка следования целей зависит скорость работы ЭС, а порой и ее работоспособность.

Математическая модель Пролога основана на теории исчисления предикатов, в частности, на процедурной интерпретации Хорновых дизъюнктов, содержащих не более одного заключения, Роберта Ковальского

из Эдинбурга. Алан Колмероэ, автор языка Пролог, начал работы над полноценной компьютерной реализацией трудов Ковальского с 1972 года во французском университете Марсель-Экс. Он составил алгоритм формального способа интерпретации процесса логического вывода и разработал систему автоматического доказательства теорем, которая была написана на Фортране. Она-то и послужила прообразом Пролога. Название его произошло от Programmation en Loqicue - ЛОГическое ПРОграммирование. Вскоре появились первые компиляторы с этого языка, в частности, прекрасная реализация Дэвида Уоррена для компьютера DEC-10 в Эдинбурге, ставшая своего рода стандартом вплоть до сегодняшнего дня. Эффективность этой версии заставила специалистов по искусственному интеллекту по-новому взглянуть на Пролог. В некоторых приложениях, типичных для Лиспа, таких как обработка списков, Пролог уже не уступал своему конкуренту, что и послужило в дальнейшем стимулом для ряда специалистов по логическому программированию к переходу на этот язык.

В качестве типовых данных Пролог использует элементарные единицы данных, так называемые атомы - строки символов и числа. Из атомов составляются списки и бинарные деревья. Сама "программа" строится из последовательности фактов и правил, и затем формулируется утверждение, которое Пролог будет пытаться доказать с помощью введенных правил. Таким способом можно описывать очень сложные проблемы, которые будут решаться самим Прологом *автоматически*. Это происходит с помощью метода сопоставления и рекурсивного поиска. Рекурсия играет в Прологе большую роль.

С распространением Пролога появилась возможность создавать интеллектуальные нереляционные базы знаний с иерархической структурой на основе стандартного механизма с гибкой организацией очень сложных запросов. Были написаны эффективные программы для решения переборных задач, в частности, из области молекулярной биологии и проектирования СБИС, где требовалось учитывать либо сложные внутренние структуры,

либо большое число правил, описывающих организацию объекта. Пролог хорошо зарекомендовал себя в качестве экспертной оболочки и при решении задач грамматического разбора. Что весьма характерно, первый высокопроизводительный компилятор этого языка (Эдинбургская версия) был написан на самом Прологе. И немудрено, ведь все формальные синтаксические описания грамматик в Бэкус-форме прекрасно записываются в терминах Пролога.

Долгое время среди разработчиков этого языка шла напряженная борьба между сторонниками оригинальной семантики Пролога и специалистами, стремившимися пожертвовать ясной структурой языка ради повышения эффективности реализации. В частности, стала играть роль последовательность правил в базе данных. Дело в том, что нередко для получения быстрого ответа оптимально использовать сначала, например, наиболее простые правила, или наиболее эффективные с точки зрения человека. Программа на Прологе постепенно стала приближаться к обычным процедурным - последовательным языкам. Немалую роль в этом сыграло и искусственно введенное понятие отсечения, своего рода аналог столь нелюбимого Дейкстрой goto. Теперь программист мог по своему усмотрению динамически отсекать бесплодные, по его мнению, ветви деревьев перебора, что приводило к многократному (на два-три порядка) повышению скорости работы программ, но при этом сильно нарушалась ясность ее структуры и возникало множество проблем, связанных с отладкой.

К счастью, развитие вычислительной техники в сочетании с уникальной структурой языка дало свои результаты. При появлении первых параллельных компьютеров люди, программирующие на Прологе, быстро осознали пагубность различных "нововведений" типа оператора отсечения, лишавших язык оригинальной чистоты, и вернулись к первоначальной версии языка. Пресловутая последовательность правил перестала играть роль, так как появилась возможность вычислять их параллельно, а в силу того, что математическая теория Пролога не накладывает никаких

требований на упорядоченность фактов и правил в базе, то скорость работы программы стала линейно пропорциональной числу процессоров. Имеются бесплатные и коммерческие версии Пролога для реализаций на параллельных компьютерах, например, Densitron CS Prolog для транспьютеров, или Paralogic. В японском проекте компьютера пятого поколения все программное обеспечение создавалось на Пролог подобном языке.

V МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Лабораторные работы проводятся по подгруппам в компьютерном классе. Каждый студент получает задание к лабораторной работе.

При подготовке к лабораторным работам рекомендуется использовать следующую учебную литературу:

1. Малпас Дж. Реляционный язык Пролог и его применение. – М., Мир, 1990
2. Янсон А. Турбо – Пролог в сжатом изложении. – М., Мир, 1995
3. Братко И. Программирование на языке Пролог для искусственного интеллекта. / Пер. с англ.М.: Мир, 1990.
4. Стерлинг Л., Шапиро З. Искусство программирования на языке Пролог. – М., Мир, 1990
5. Доорс Дж., Рейблейн А.Р., Вадера С. Пролог – язык программирования будущего. – М., Финансы и статистика, 1990
6. Кларк К., Маккейб Ф. Введение в логическое программирование на Микро – Прологе. – М., Радио и связь, 1987

Выполняя задание, студент пользуется теоретическим материалом приведенном перед заданием к лабораторной работе и учебной литературой.

Выполненную лабораторную работу студент должен продемонстрировать преподавателю на экране компьютера и ответить на контрольные вопросы.

VI ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

ЛАБОРАТОРНАЯ РАБОТА № 1.

Тема: ИЗУЧЕНИЕ РАБОТЫ С ИНТЕГРИРОВАННОЙ ОБОЛОЧКОЙ СИСТЕМЫ ТУРБО ПРОЛОГ.

Цель: Освоение основных режимов работы в интегрированной оболочке системы Турбо Пролог: редактирования, компиляции, отладки, ведения диалога и работы с файлами.

Общие сведения

1. Турбо-Пролог, версия 2.0

Система Турбо Пролог версия 2.0 может работать на ПЭВМ, совместимых с IBM PC XT/AT и PS/2, с ОЗУ минимум 384 Кбайт и двумя НГМД по 360 Кбайт. Рекомендуется иметь ОЗУ 512/640 Кбайт и НМД типа "Винчестер". В файле config.sys должно быть указано files=20 buffers=40.

Программа на Турбо Прологе состоит из следующих в определенном порядке секций и имеет следующую структуру [2]:

```
constants    /* Секция объявления констант. Может отсутствовать */
 domains     /* Секция объявления нестандартных и/или составных типов данных. Может отсутствовать */
database - имя_ВБД  /* Необязательная секция объявления предикатов для работы с внутренней базой данных (ВБД) */
predicates   /* Секция объявления предикатов */
clauses      /* Секция объявления правил и фактов */
goal         /* Секция объявления внутренней цели. Может отсутствовать */
```

При составлении программы на Прологе необходимо соблюдать следующие ограничения:

- комментарии в программе могут располагаться в программе на любом месте. Комментарий начинается либо с символа % либо с последовательности символов /* и заканчиваться */;

- в программе может использоваться только один раз секция GOAL;

- все предикаты в CLAUSES с одинаковыми именами должны записываться подряд;

- большинство стандартных предикатов выполняют несколько функций в зависимости от состояния параметров, входящих в предикат. Известные параметры называют входными (INPUT – (i)), неизвестные – выходными (OUTPUT – (o)). Совокупность входных параметров определяет работу предиката. Эта совокупность называется проточным шаблоном.

1.1. Интегрированная оболочка системы Турбо-Пролог предоставляет следующие возможности:

- создавать и редактировать тексты программ;
- выполнять и отлаживать программы;
- транслировать программы в объектные файлы;
- компоновать объектные файлы в исполняемые модули;
- получать справочную информацию, изменять размеры окон и их цвет;
- устанавливать параметры и конфигурацию системы.

При первоначальном входе в интегрированную среду Турбо-Пролога на экране монитора появляется главное меню (рис.1). В верхней строке находятся названия 6 основных режимов работы системы. Текущее положение в меню отмечено выделяющейся по цвету и яркости прямоугольной полоской. Перемещая эту полоску (курсор) с помощью клавиш с горизонтальными стрелками нажатием клавиши Enter можно выбрать необходимый режим. Это можно сделать также одновременным нажатием клавиши Alt и первой буквы названия соответствующего меню, например, для выбора режима редактирования достаточно нажать Alt-E.

Для удобства работы для наиболее часто используемых операций в оболочке Турбо Пролога вместо выбора из меню (или подменю) можно использовать нажатие функциональных клавиш, либо определенного сочетания клавиш (Hot keys). Действие той или иной функциональной клавиши может быть различным в зависимости от того, в каком режиме

находится система. Более полную подсказку можно получить нажатием клавиш Alt-H.

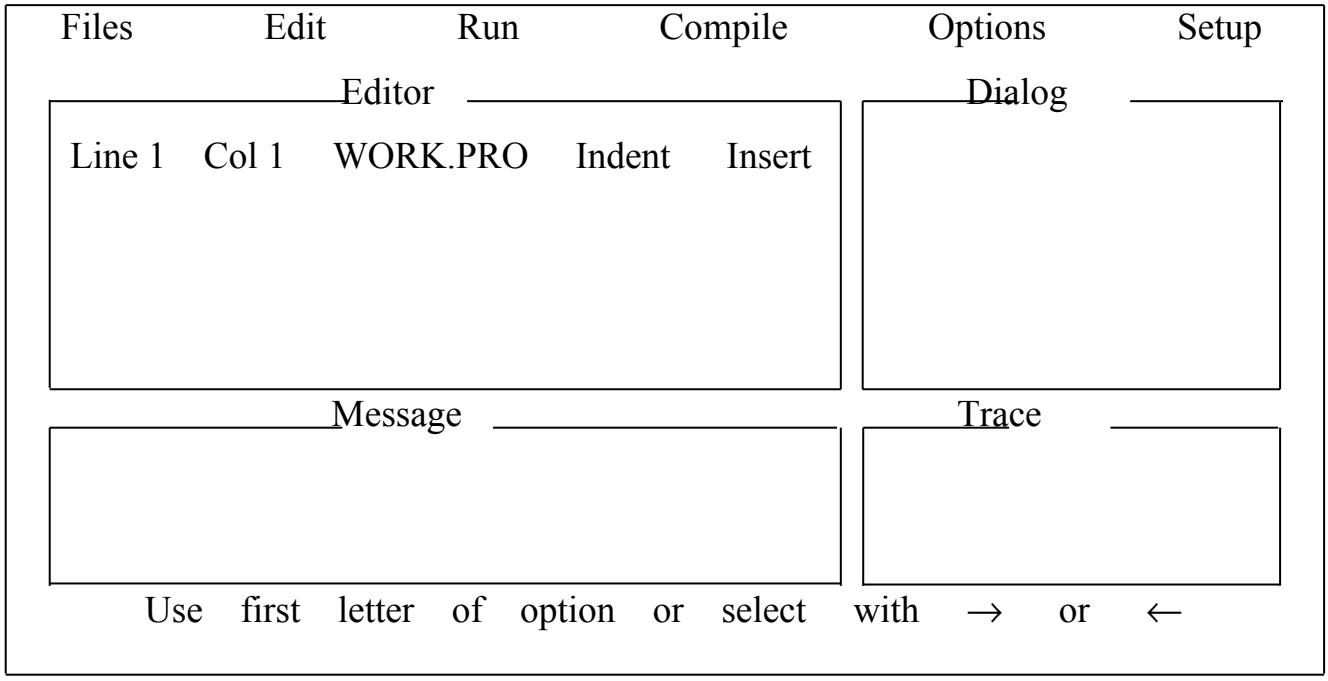


Рис. 1

Экран разделен на 4 окна:

- окно редактора текстов, в которое загружаются отлаживаемые или редактируемые программы;
- окно диалога является по умолчанию окном, если в программе не назначены другие, ввода и вывода из программы;
- окно для вывода сообщений системы;
- окно сообщений отладчика (трассировки).

Первоначальное разделение экрана может быть изменено в режиме Setup. Окно, в котором находится курсор, называется текущим окном.

Нижняя строка является строкой подсказки или иначе навигационной строкой. Навигационную строку вверху, показывающую местоположение курсора (Line - номер текущей строки, Col - позиция курсора в строке), название редактируемого файла и режимы редактирования, имеет также окно редактирования. Если имя редактируемого фала не задано, то по умолчанию он называется WORK.PRO. Если файл с таким именем находится

в текущей директории, то он автоматически загружается в окно редактирования.

Все режимы главного меню, кроме Edit и Run, содержат дополнительные подменю. Выход из любого подменю осуществляется нажатием клавиши Esc.

1.2. Цикл разработки программы

Турбо Пролог поддерживает в широком смысле структурное программирование и сопровождение проекта. Цикл разработки программы (без постановочной части) можно представить следующей схемой:

1. С помощью встроенного редактора текстов исходный текст программы вводится в систему.

2. Периодически, по мере ввода новых предикатов, программа транслируется для выявления синтаксических ошибок, которые тут же устраняются.

3. С помощью встроенных средств трассировки производится отладка каждого нового предиката.

4. Периодически производится структуризация программы. Независимые части программы выделяются в отдельные модули.

5. Процесс повторяется до получения программы, удовлетворяющей внешним спецификациям, которые, в частности, сами могут быть написаны на Турбо Прологе.

2. Основные режимы работы

Основные режимы работы [2] заданы главным меню в верхней строке экрана. Такими режимами являются:

Files - работа с файлами и взаимодействие с MS-DOS

Edit - ввод и редактирование программы

Run - выполнение и отладка программы

Compile - трансляция и компоновка программы

Options - установка режимов компиляции

Setup - установка режимов работы оболочки

2.1. Работа с файлами и взаимодействие с MS-DOS

При выборе режима Files в верхней части экрана под словом Files появится подменю (рис. 2.).

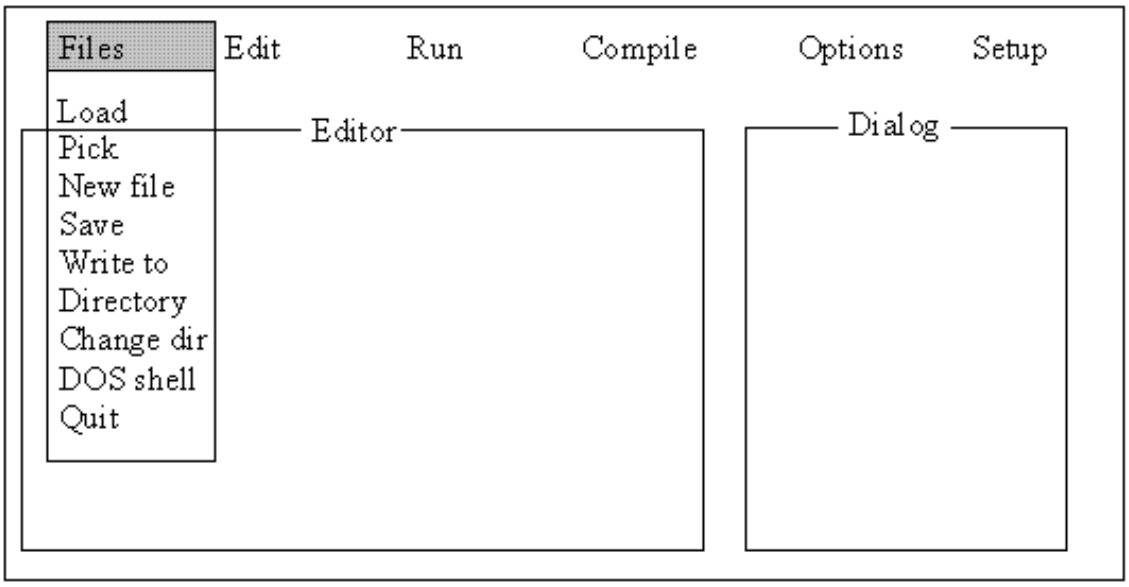


Рис. 2

Перемещение по подменю осуществляется также с помощью клавиш со стрелками, вводом первой (заглавной) буквы слова. Для быстрого выполнения наиболее частых операций имеются зарезервированные функциональные клавиши или одновременное нажатие нескольких клавиш (Hot keys).

Load – позволяет загрузить исходный текст программы на Прологе в окно редактора для его последующей модификации, выполнения или отладки.

Pick – позволяет работать из оболочки Турбо-Пролога одновременно с 8-ю файлами.

New file – очистка окна редактора текстов для ввода нового файла.

Save – сохранение редактируемой программы в файле на диске.

Write to – запись редактируемой программы в заданный файл на диске.

Directory – вывод содержимого указанной директории на экран.

Change dir – смена текущей директории. Равносильна команде ChDir в MS DOS.

OS shell – выполнение команды операционной системы. Выход из системы Турбо-Пролог в MS-DOS с возвратом по команде EXIT.

Quit – окончание работы. Выход из системы Турбо-Пролог в MS-DOS.

При работе с файлами приняты следующие соглашения об их типах:

.PRO - файл с исходным текстом программы

.BAK - файл сохранения после редактирования

.PRJ - файл с именами модулей, входящих в проект разрабатываемой системы

.EXE - исполняемый файл после компоновки

.OBJ - объектный файл после трансляции

.ARC - упакованный файл архивации

2.2. Ввод и редактирование программы

Встроенный редактор интегрированной оболочки Турбо-Пролога имеет набор команд, который является подмножеством команд широко известного редактора текстов WordStar. Кроме того, при редактировании можно пользоваться более короткими командами, пользуясь функциональными и управляющими клавишами.

Все команды редактирования можно переопределить, однако учитывая, что они совпадают с оболочками языков Турбо Паскаль, Турбо Си и Турбо Бейсик, это не рекомендуется делать.

Как и при работе с обычным текстовым редактором, ввод может происходить в режиме вставки текста или в режиме наложения его на уже существующий. Переход с одного режима в другой осуществляется нажатием клавиши Ins.

В случае слияния файлов, копирования блока из одного файла в другой и т.п. в системе Турбо Пролог может быть открыто дополнительное окно редактирования (Aux Editor) в нижнем правом углу экрана.

Команды редактора [2] условно делятся на команды перемещения, команды удаления текста, команды работы с блоками текста и

вспомогательные. Выход из редактора в главное меню - F10 или по одновременному нажатию клавиши Alt и клавиши с первой буквой имени соответствующего режима.

Команды перемещения

Направление перемещения	Клавиши
Курсор вправо на один символ	Стрелка вправо или Ctrl-D
Курсор влево на один символ	Стрелка влево или Ctrl-S
Курсор вверх на одну строку	Стрелка вверх или Ctrl-E
Курсор вниз на одну строку	Стрелка вниз или Ctrl-X
Курсор вправо на одно слово	Ctrl-Стрелка вправо
Курсор влево на одно слово	Ctrl-Стрелка влево
Курсор в конец строки	End
Курсор в начало строки	Home
Курсор в начало окна	Ctrl-Home
Курсор в конец окна	Ctrl-End
Курсор вверх на один экран	PgUp или Ctrl-R
Курсор вниз на один экран	PgDn или Ctrl-C
Курсор в начало файла	Ctrl-PgUp или Ctrl-Q R
Курсор в конец файла	Ctrl-PgDn или Ctrl-Q C

Команды удаления текста

Название команды	Клавиши
Удаление символа над курсором	Del
Удаление символа слева от курсора	BackSpace или Ctrl-H
Удаление слова над курсором	Ctrl-T
Удаление строки над курсором	Ctrl-Y
Удаление начала строки до курсора	Ctrl-Q T
Удаление от курсора до конца строки	Ctrl-Q Y

Если при редактировании программы возникли какие-либо трудности, то нажатие клавиши F1 вызовет следующее меню системы подсказки:

Edit Help – [Помощь при редактировании] Заголовок меню

Show help file – вывод на экран файла справочника по всей системе. Нажатие F5 позволяет увеличить окно со справочником до размеров экрана.

Повторное нажатие уменьшит его до прежних размеров. Выход из справочника по Esc.

Cursor movement – вывод на экран справочника по командам для перемещения курсора.

Insert & Delete – справочник по операциям вставки и удаления текста.

Block Functions – справочник по операциям над блоками текста.

WordStar-like – справочник по командам редактирования, аналогичным командам редактора текстов WordStar.

Miscellaneous – справочник по командам редактирования не входящим в какую-либо группу.

Global functions – справочник по командам, выполняющим операции над всем текстом.

Hot keys – справочник по клавишам для быстрого выполнения наиболее часто встречающихся операций. При этом выбранная с помощью клавиш со стрелками операция может быть тут же выполнена.

2.3. Выполнение и отладка программ

Режим Run обеспечивает компиляцию находящейся в окне редактора текстов программы и ее выполнение. Этот режим обеспечивает быструю интерактивную отладку программы, так как при обнаружении синтаксических и семантических ошибок имеется возможность их тут же исправить. Мощным инструментом отладки программ в системе являются встроенные средства трассировки.

Возможны следующие виды трассировки:

- пошаговая трассировка всей программы
- трассировка заданных предикатов
- трассировка в режиме оптимизации
- интерактивное включение/выключение трассировки
- интерактивный вывод результатов трассировки и выходных сообщений на печать и в дисковый файл.

Отметим, что трассировка является также важным средством изучения Турбо Пролога, так как, в случае непонимания работы какого-либо предиката, его работу можно посмотреть по шагам.

По умолчанию режим трассировки выключен (Off). Для установки/отключения режима трассировки следует выбирать подменю "Директивы компилятора" меню Options (См. п. 2.5) и в нем выбрать вход Trace. На экране появится небольшое меню, содержащее три входа:

Trace – включить режим полной трассировки;

ShortTrace – включить режим сокращенной трассировки;

Off – выключить трассировку.

Если выбрать Trace (выход из всех подменю по Esc), то после запуска программы на выполнение (Run или Alt-R) в окне диалога появится подсказка Goal:. После ввода цели, в окне трассировки за словом CALL: появляется вызываемый предикат.

Нажатие клавиши F10 разрешает выполнение следующего шага программы. Одновременно в окне редактирования курсор показывает на выполняемый в текущий момент предикат.

После подсказки REDO: отображаются результаты вычисления (унификации).

При успешном поиске подстановке после слова RETURN: выводится ее результат. При неудаче выводится подсказка FAIL.

Трассировка в любой момент может быть прекращена нажатием клавиши Esc.

Нажатие Alt-T позволяет изменить режим трассировки. При этом на экране появляется следующее меню:

Status	On	Запрет/разрешение отображения статуса
Trace window	On	Запрет/разрешение окна трассировки
Edit window	On	Запрет/разрешение окна редактирования

После нажатия Shift-F10 можно по желанию изменить размеры окон.

Внутри программы можно размещать предикат trace(on), который включает режим трассировки, или trace(off) - выключающий ее.

Нажатие клавиш Alt-P позволяет перенаправить вывод результатов трассировки на печать или в файл PROLOG.LOG на диске.

Режим ShortTrace отличается от Trace только тем, что компилятор производит оптимизацию программы (в режиме Trace оптимизация не производится). Это особенно важно при отладке рекурсивных вызовов, особенно при "хвостовой рекурсии", когда рекурсивный вызов является последней подцелью в теле предиката. Этот вид рекурсивных вызовов оптимизируется Турбо Прологом и преобразуется в цикл, что экономит память при исполнении.

2.4. Трансляция и компоновка программ

Подменю режима Compile содержит следующие входы:

Memory – отладочная трансляция в память

OBJ file – трансляция в файл типа .OBJ

EXE file (auto link) – трансляция с созданием исполняемого (.EXE) файла с автоматической компоновкой модулей

Project (all modules) – трансляция всех входящих в проект модулей;

Link only – выполнение компоновки модулей в исполняемый файл.

2.5. Установка режимов компиляции

Исходный текст программы на Турбо Прологе перед выполнением проходит обязательную фазу компиляции и компоновки. Для определения параметров этих процессов служит режим Options.

Меню режима Options содержит следующие входы:

Link options (Опции компоновки)

Содержит подменю:

Map file ON/OFF – создавать или нет файл с картой компоновки.

Libraries – ввод имен пользовательских библиотек для компоновки.

Edit PRJ file (Редактирование файла с описанием проекта)

Запрашивается имя файла, в котором перечислены модули, входящие в проект. Этот файл затем используется компоновщиком.

Compiler Directive (Директивы компиляции)

Содержит подменю (названия входов даны в круглых скобках) позволяющие установить:

- распределение памяти (Memory allocation) под код, стек, тип и рекурсию. Размер этих областей устанавливается в параграфах (параграф равен 16 байтам);
- какие виды контроля выполнять во время исполнения программы (Run-time check): нажатие клавиши Breack, нарушение границ стека и целочисленное переполнение;
- допустимый уровень ошибок трансляции (Error level): ошибки недопустимы, по умолчанию (1), максимальный (2);
- выдачу предупреждения при наличии недетерминированных предикатов (Non-determ warning ON/OFF);
- предупреждение о наличии переменных, которые используются в предикате только один раз (Variable used once warning ON/OFF);
- уровень трассировки (Trace): полная, сокращенная и трассировка выключена;
- включение вывода диагностики по результатам трансляции (Diagnostics ON/OFF).

2.6. Получение справочной информации, изменение размеров окон и их цвета

Подменю режима SetUp содержит следующие входы:

Colors - изменение цвета окон при наличии цветного монитора. На экран выводится таблица цветовой гаммы, из которой для каждого окна выбирается цвет фона и цвет изображения;

Window size - изменение размеров окон. При выборе этой опции в нижней строке экрана появляется подсказка, как с помощью клавиш со стрелками изменить размер или местоположение окна.

Directories - установка директорий по умолчанию.

Miscellaneous - разные опции связанные с настройкой на адаптер видеомонитора;

Load SYS file - загрузка файла с параметрами интегрированной среды;

Save SYS file сохранение параметров настройка среды интегрированной среды в файле типа *.SYS.

3. Стандартные предикаты

3.1. Общесистемные стандартные предикаты

В этом разделе приведены предикаты [2], позволяющие использовать возможности предоставляемые операционной системой MS DOS.

- date (Год,Месяц,День) (integer, integer, integer): прототип (o,o,o) – считывает системную дату, прототип: (i,i,i) – установить дату.

Например:

date(J,M,T) - результат: J=2001,M=11,T=01

date(2001,02,26) – системная дата устанавливается на 26.02.2001.

- time(Час, Минуты, Секунды, Сотые_секунды) (integer, integer, integer, integer): прототип (o,o,o,o) - связывает соответствующие параметры с текущим временем, прототип (i,i,i,i) - устанавливает системное время.

Например:

time(S,M,Sek,_) - результат: S=15,M=35,Sek=22

time(17,05,0,0) - часы будут поставлены на 17:05:00.

- system(Строка_с_командой_DOS) (string): прототип (i) - заданная строка должна быть допустимой командой DOS. Разрешаются встроенные и внешние команды DOS, такие как файлы с расширением ".BAT". Заданная команда выполняется. Предикат system не выполняется, если команда некорректна с точки зрения DOS. Если "Строка_с_командой_DOS" пустая (""), то COMMAND.COM активируется в интерактивном режиме.

Например:

system("dir A:") - выводится каталог накопителя A.

- comline(Строка) (string): прототип (o) - читает параметры, которые заданы в команде вызова скомпилированной программы на Турбо – Прологе.

Например:

TEST abc (вызов в MS - DOS) - если в программе с именем TEST стоит команда comline(X), то строка abc будет связана с X.

- beep - производит звуковой сигнал
- bios(Номер_прерывания,Регистры_входные,Регистры_выходные)
- (integer, reg, reg): прототип (i,i,o) - обеспечивает формирование прерывания с заданным номером. Регистры получают значения, установленные параметром "Регистры_входные". После обработки прерывания содержимое регистров связывается с параметром "Регистры_выходные". Параметры "Регистры_входные" и "Регистры_выходные" должны принадлежать домену regdom, который определяется следующим образом:

regdom = reg(AX, BX, CX, DX, SI, DI, DS, ES), где все аргументы имеют тип integer.

Например: bios($21, reg(AX,0,0,0,0,0,0,0),reg(Nr,_,_,_,_,_,_,_)) - производится чтение с текущего накопителя. Номер накопителя связывается с Nr.

3.2. Предикаты преобразования типов

- char_int(Символ,Число) (char,integer): прототип (i,o) - связывает параметр "Число" с кодом ASCII параметра "Символ"; прототип (o,i) - связывает параметр "Символ" с символом, код которого определяется параметром "Число"; прототип (i,i) - выполняется успешно, если код, определяемый параметром "Число", является ASCII – кодом символа, определяемого параметром "Символ".

Например:

char_int('A',X) - переменная X принимает значение 65.

char_int(X,66) - переменная X принимает значение ‘B’.

char_int(‘A’,65) - выполняется успешно.

- str_char(Строка,Символ) (string,char): прототип: (i,o) - заданная первым параметром строка, состоящая из единственного символа, преобразуется в символ. Символ связывается со вторым параметром; прототип (o,i) - преобразуется символ в строку, состоящую из единственного символа и связывает ее с заданной переменной; прототип: (i,i) - выполняется успешно, если параметры связаны с представлениями одного и того же символа.

  Например:

str_char(“A”,X) - результат X=’A’.

str_char(X,’A’) - результат X=”A”.

str_char(“A”,‘A’) - выполняется успешно.

- str_int(Строка,Целое число) (string,integer): прототип:(i,o) - преобразует строку в целое число. Число связывается со вторым параметром; прототип (o,i) - связывает с первым параметром строку, представляющую собой запись целого числа, связанного со вторым параметром; прототип(i,i) - выполняется успешно, если связанная с первым параметром строка является представлением числа, связанного со вторым параметром.

  Например:

str_int(“123”,X) - результат: X=123

str_int(X,456) - результат: X=”456”

str_int(“234”,234) - выполняется успешно.

- str_real(Строка,Действительное число) (string,real): прототип:(i,o) - связывает второй параметр с действительным числом, определяемым записью числа в строке, заданной первым параметром; прототип (o,i) - связывает с первым параметром строку, представляющую собой запись действительного числа, заданного

вторым параметром; прототип:(i,i) - выполняется успешно, если связанная с первым параметром строка является представлением числа, связанного со вторым параметром.

Например:

str_real(“1.23”,X) - результат: X=1.23

str_real(X,0.56) - результат: X=”0.56”

str_real(“4.567”,4.567) - выполняется успешно.

- file_str(Имя_ файла_ DOS,Строка) (string,string): прототип (i,o) - читает строку из заданного файла и связывает ее с параметром "Строка". Максимально допустимый размер строки – 64 К. Признаком конца строки является символ Ctrl – Z (десятичный код ASCII=26).

Например: file_str(“B:TEXT1”,X) - символы из файла TEXT1 на накопителе B будут прочитаны и связаны с переменной X.

- field_str(Строка, Столбец, Длина, Строка_символов) (integer, integer, integer, string): прототип (i,i,i,i) - записывает строку, связанную с параметром "Строка_символов", в поле, определяемое длинной и номерами строки и столбца. Если строка длиннее, чем заданное поле, то записывается только начало строки. Если строка короче, то оставшиеся позиции поля заполняются пробелами; прототип (i,i,i,o) - строка, определяемая длиной и позицией, связывается с параметром "Строка_символов". Проследите, чтобы поле в текущем окне соответствовало параметрам.

Например:

field_str(15,5,5,"hollo") - строка "hollo" записывается в поле, начинающееся с позиции (15,5).

field_str(10,30,5,X) - строка длиной 5, начинающаяся с позиции (10,30), связывается с переменной X.

- str_len(Строка,Длина) (string,integer): прототип (i,o) - с параметром "Длина" связывается количество символов в заданной строке; прототип (i,i) - выполняется успешно, если строка имеет заданную длину.

Например:

str_len("hollo", X) - результат: X=5.

str_len("book", 4) - выполняется успешно.

- isname(Строка) (string) прототип (i) - выполняется успешно, если последовательность символов, связанная с параметром, представляет собой имя, допустимое в Турбо – Прологе.

Например:

isname ("abcd") - выполняется успешно.

- upper_lower(Строка1,Строка2) (string, string): прототип (i,i) - выполняется успешно, если с первым и вторым параметром связаны идентичные строки, представленные соответственно прописными и строчными буквами; прототип (i,o) - связывает со вторым параметром строку, полученную из строки, связанной с первым параметром, заменой прописных букв на строчные; прототип (o,i) - связывает с первым параметром строку, полученную из строки, связанной со вторым параметром, заменой строчных букв на прописные.

Например:

upper_lower("A","a") - выполняется успешно.

upper_lower("ZDF",X) - результат: X="zdf"

upper_lower(X,"house") - результат: X="HOUSE"

3.2. Арифметические операции

+ сложение

- вычитание

* умножение

/ деление

mod абсолютная величина

div целочисленное деление

3.3. Операторы отношений

Операторы отношений являются инфиксными операторами (т.е. должны находится между двумя сравниваемыми величинами). Свободные переменные в операторах отношений не допускаются. Для операторов отношений приняты следующие обозначения:

< меньше; > больше; = равно; <= меньше или равно; >= больше равно; <> не равно.

3.4. *Математические функции*

Функция	**Описание**
abs(X) /* (Var) (i) */	Возвращает абсолютное значение X
round(X) /* (Var) (i) */	Возвращает округленное целое значение X
sqrt(X) /* (Var) (i) */	Возвращает квадратный корень из X
trunc(X) /* (Var) (i) */	Возвращает целое значение X отбрасывая дробную часть
exp(X) /* (Var) (i) */	Возвращает значение e в степени X
log(X) /* (Var) (i) */	Возвращает десятичный логарифм X
ln(X) /* (Var) (i) */	Возвращает натуральный логарифм X
arctan(X) /* (Radians) (i) */	Возвращает арктангенс X
cos(X) /* (Radians) (i) */	Возвращает косинус X
sin(X) /* (Radians) (i) */	Возвращает синус X
tan(X) /* (Radians) (i) */	Возвращает тангенс X

Все тригонометрические функции требуют, чтобы аргумент X задавался в радианах.

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ № 1

Тема: Создание отношений в Прологе.

Последовательность действий

1. Изучить пояснения к работе.

2. Загрузить Турбо Пролог. На экране появится главное меню и окна, изображенные на рис. 1.

3. Нажимая клавишу с левой или правой горизонтальной стрелкой последовательно пройти по главному меню. Обратите внимание на появление подменю по мере перехода от одного режима к другому. Войти в подменю Files. С помощью клавиш с вертикальными стрелками переместить прямоугольную полоску на слово Directory. Нажать клавишу Enter (/вв/). На экран будет выведено содержание текущей директории.

4. Сделать текущим окно редактирования. Вернуться в главное меню нажатием клавиши F10.

5. Войдите в режим редактирования (Alt-E), изучите действие клавиши F1 (меню подсказок) и введите следующие определения:

```
domains                    /вв/
  fakt = symbol            /вв/
predicates                 /вв/
  male(fakt)               /вв/
  female(fakt)             /вв/
  mother(fakt,fakt)        /вв/
  father(fakt,fakt)        /вв/
  wife(fakt,fakt)          /вв/
clauses                          /вв/
  male(“саша”).                  /вв/
  female(“марина”).              /вв/
  mother(“наташа”,”саша”).       /вв/
  father(“петя”,”коля”).         /вв/
  wife(“наташа”,”петя”).         /вв/
  ……
```

6. Введите в секцию clauses 5-7 фактов для предикатов male, female, mother, father, wife.

7. Пользуясь средствами Турбо Пролога постройте предикаты для выражения следующих связей между объектами:

```
son(X,Y)              /вв/
```

daughter(X,Y) /вв/

brother(X,Y) /вв/

grandmother(X,Y /вв/

grandfather (X,Y) /вв/

cusins(X,Y) /вв/

uncle(X,Y) /вв/

aunt(X,Y) /вв/

Например:

parent(X,Y):- father(X,Y);mother(X,Y). /вв/

husbend(X,Y):-wife(Y,X). /вв/

8. После ввода каждого нового предиката, программу следует откомпилировать, для чего нажать клавишу F9. Если при компиляции будут обнаружены синтаксические ошибки, то их следует исправить и добиться безошибочной компиляции (см. п. 2.2).

9. Если компиляция прошла успешно, перейти в режим исполнения: Run в главном меню или нажатие клавиш Alt-R.

10. В режиме исполнения каждый новый введенный вами предикат проверьте задавая в окне диалога после подсказки "Goal:" внешнюю цель с этим предикатом.

Например:

son(X,Y) /вв/

после ввода соответствующего предиката.

11. Включить режим трассировки и просмотреть выполнение предиката по шагам (см. п. 2.3), нажимая клавишу F10 для выполнения каждого следующего шага. В окне трассировки будет показано связывание переменных и результат вычисления предиката.

12. Используя меню Setup, изменить размеры окон редактирования и трассировки.

13. По окончанию работы выйдите из системы выбрав Quit в меню Files или нажав клавиши Alt-X.

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ № 2

Тема: Использование рекурсии.

Приведенная ниже база данных «путешествие» содержит факты, каждый из которых имеет по четыре аргумента. Каждый факт устанавливает, что можно совершить путешествие на транспортных средствах некоторой компании (аргумент 1) из одного города (аргумент 2 – пункт отправления) в другой город (аргумент 2 – пункт назначения) и при этом воспользоваться некоторым видом транспорта (аргумент 4).

% компания отправл. прибытие вид трансп.

путешествие(круизавиа, благовещенск, москва, самолет).
путешествие(автосервис, благовещенск, белогорск, автобус).
путешествие(ж/дсервис, благовещенск, красноярск, поезд).
путешествие(круизавиа, москва, сочи, самолет).
путешествие(авиалинии, красноярск, брест, самолет).
путешествие(ж/дсервис, тында, владивосток, поезд).

1. Правило «можно_путешествовать» устанавливает косвенную связь между городами, которая будет соблюдаться в том случае, если возможно путешествие из одного города в другой через третий – промежуточный – город. Декларативная трактовка данного правила может быть такой:

Будет возможным совершить путешествие между городами «Город А» и «Город Б», если можно добраться из города «Город А» в промежуточный пункт «Город Б» и можно добраться из города «Город Б» в «Город В».

или

путешествие из города «Город А» в «Город В» будет возможным, если либо 1) существует прямая транспортная связь между этими городами, либо 2) можно совершить путешествие из города «Город А» в некоторый промежуточный пункт «Город Б»Б а затем добраться из города «Город Б» в «Город В».

2. Правило «конкурент» гласит, что любые две транспортные компании будут конкурентами, если обе они обслуживают один и тот же маршрут. Декларативная трактовка данного правила может быть такой:

«Компания 1» будет конкурентом для компании «Компания 2», если существуют два города «Город А» и «Город Б», такие, что «Компания 1» обеспечивает перевозки между городами «Город А» и «Город Б», и «Компания 2» обеспечивает перевозки между городами «Город А» и «Город Б».

Последовательность действий

Изучить пояснения к работе.

Воспользуйтесь редактором для создания файла для работы с приведенными фактами и правилами.

Добавьте в программу 10 - 15 фактов для предиката "путешествие" и промоделируйте отношения, включая все необходимые факты и правила. Протестируйте Вашу программу, чтобы проверить, выполняет ли она то, что было задумано.

Составьте запросы, позволяющие:

a) получить сведения обо всех парах компаний - конкурентов;
b) получить сведения о возможности совершить путешествие из одного города в другой с выводом всех городов через которые прошел маршрут путешествия от исходного пункта к месту назначения.

Получить один ответ на запрос в следующем виде, например:

М=благовещенск --- самолет → москва --- поезд → смоленск

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ № 3

Тема: Работа со списками.

1. Создать случайным образом список состоящий из K нулей и единиц.

2. Написать предикат PL(L+,N-) – истинный тогда и только тогда, когда N – предпоследний элемент списка L, имеющего не менее двух элементов.

3. Определите возведение в целую степень через умножение и деление.

4. Вставить подсписок в определенное место списка.

5. Удалить все заданные элементы из списка.

6. Сложить два списка.

7. Напишите предикат subst(+V, +X, +Y, -L) – истинный тогда и только тогда, когда список L получается после взаимной замены X на Y, т.е. X → Y, Y → X.

8. Напишите предикат, который определяет, является ли данное натуральное число простым.

9. Напишите предикат p(+N, +K, -L) – истинный тогда и только тогда, когда L – список всех последовательностей (списков) длины K из чисел 1, 2, …, N.

10. Напишите предикат p(+N, -L) – истинный тогда и только тогда, когда список L содержит все последовательности (списки) из N нулей и единиц, в которых никакая цифра не повторяется три раза подряд (нет куска вида XXX).

ЛАБОРАТОРНАЯ РАБОТА № 2.

РАБОТА С ВНУТРЕННЕЙ И ВНЕШНЕЙ БАЗАМИ ДАННЫХ СИСТЕМЫ ТУРБО ПРОЛОГ

Цель: Освоение основных режимов работы с внутренней и внешними базами данных в системе Турбо Пролог: создание базы данных, занесение в нее фактов, поиск в БД, удаление записей, удаление БД.

Общие сведения

1. Концепция работы с файлами

Турбо Пролог поддерживает работы с файлами, а также работу с внутренней и внешней базами данных. Одновременно в системе можно

производить операции с двумя файлами (устройствами): одним входным и одним выходным. Файлы объявляются в секции domains следующим образом:

domains

file = name _file

Например:

domains

file = textfile или file = my_file

В системе имеются предопределенные файлы с именами: keyboard (клавиатура), printer (устройство печати), com1 (коммуникационный порт), screen (экран), stdin (стандартное устройство ввода), stdout (стандартное устройство вывода), stderr (стандартное устройство для сообщений об ошибках). По умолчанию stdin и stdout открыты и назначены соответственно на keyboad и screen.

Набор предикатов для ввода ориентирован на простые типы данных и включает в себя:

1) предикаты для работы с клавиатуры:

- inkey(Символ) (char): прототип (o) - читает символ со стандартного устройства ввода, если он доступен, иначе inkey считается невычисленным (fail).

Например:

```
clauses
  ready(X):-inkey(Y),X=Y,!.
```

При трассировке предикатов с inkey рекомендуется с помощью клавиш Alt-T установить Edit window в режим Off.

- keypressed - проверяет была ли нажата какая-либо клавиша на клавиатуре, если не была, то fail.

2) предикаты ввода данных:

- readln(Строка) (string), (o) - читает строку с текущего устройства ввода и связывает ее с заданной переменной.

Например:

readln(S)

- readterm(Область,Терм) (Name,Variable), (i,o) - читает объект, который был записан предикатом write. С помощью readterm осуществляется доступ к фактам в файле.

Например:

```
domains
    person=p(name,surname,height)
clauses
    readterm(person,p(N,S,H))
```

Файл, открытый при помощи предикатов openread и readdevice, содержит:

p("Seep","Maier",195)

Соответственно: N = Seep S = Maier H = 195.

- readint(Целочисленная_переменная) (integer), (o) - читает целое число с текущего устройства вывода и связывает его с заданной переменной.
- readreal(Переменная_вещественного_типа) (real), (o) - читает действительное число с текущего устройства ввода и связывает его с заданной переменной.
- readchar(Символьная_Переменная) (char), (o) - читает символ с текущего устройства ввода и связывает его с заданной переменной. В отличие от inkey устанавливает режим ожидания ввода.
- file_str(ИмяФайлаDOS,Строка) (string,string), (i,o) - читает строку из заданного файла и связывает ее с параметром "Строка". Максимально допустимый размер строки - 64 К. Признаком конца строки является символ Ctrl -Z (десятичный код ASCII = 26).

Например: file_str("B:TEXT1",X)

3) предикаты вывода данных:

- write(A1,A2,A3,…) (A1,A2,A3,… - константы или переменные), (i) - записывает заданные значения на текущее устройство вывода.

Наряду с константами и переменными может использоваться также и обратный слэш. Он встречается в следующих комбинациях:

\n – выдается пробел;

\t – происходит переход к следующей позиции табуляции;

\номер – код ASCII выдаваемого символа.

Например: write(“введенное имя”,name) – на экране, который обычно является текущим устройством вывода, появляется текст: “введенное имя максим”, если переменная name связана со строкой “максим”

- nl - выводит пустую строку
- writef(Формат, A1,A2,A3,…) (i) – предикат форматного вывода данных.

В строке формат после знака процента используются следующие ключи:

% g числа с плавающей точкой в наиболее компактном формате (по умолчанию).

% e числа с плавающей точкой в экспоненциальной форме.

% f числа с плавающей точкой в формате с фиксированной точкой.

Например: writef(″%5.2″,X) – результат 2.80, если X=2.8.

При работе с дисковым файлом его сначала следует открыть с указанием типа выполняемых затем действий: чтения, записи, добавления текста в конец файла или модификации содержимого файла. Это делается с помощью следующих предикатов:

- openread(Символическое_имя_файла, Имя_файла_в_DOS) (file, string), прототип (i,i) – открывает файл для чтения.

Например: openread(td,″C:text.dat″) – файл DOS text.dat на устройстве C будет открыт для чтения под именем td.

- openwrite (Символическое_имя_файла, Имя_файла_в_DOS) (file, string), прототип (i,i) – открывает файл для записи.

Например: openwrite(users,″B:us.dat″) – файл DOS us.dat на устройстве B будет открыт для записи под именем users.

- openappend(Символическое_имя_файла, Имя_файла_в_DOS) (file, string), прототип (i,i) – открывает файл для дополнения.

Например: openappend(persons,″B:person.txt″) – файл DOS person.txt на устройстве B будет открыт для дополнения под именем persons.

- openmodify(Символическое_имя_файла, Имя_файла_в_DOS) (file, string), прототип (i,i) – открывает файл для модификации (как для чтения, так и для записи).

Например: openmodify(addr,″A:ADDRESSES″) – файл DOS ADDRESSES на устройстве A будет открыт для чтения или записи под именем addr.

Если в предикате openwrite задано имя файла уже существующего на диске, то после выполнения openwrite содержимое этого файла будет стерто. Проверить наличие на диске указанного файла, особенно если файл открывается не для записи, можно до выполнения операции его открытия с помощью предиката

- existfile(Имя_файла_в_DOS) (string), прототип (i) – выполняется успешно, если заданный файл присутствует в текущем каталоге, и завершается неудачно в противном случае.

Одновременно может быть открыто несколько файлов, но только два их них могут являться текущими файлами (устройствами) ввода и вывода, которые объявляются предикатами:

- readdevice(Имя_файла) (symbol): прототип (i) – присваивает текущему устройству вводимое заданное символическое имя файла; протопит (o) – связывает с параметром ″Имя_файла″ символическое имя файла текущего устройства ввода.

Например:

readdevice(adr) – последующие команды чтения осуществляют чтение из файла *adr*.

readdevice(X) – результат X=keyboard.

- writedevice(Символическое_имя_файла) (symbol): прототип (i) - ставит в соответствие текущему устройству вывода заданное символическое имя файла; протопит (o) – связывает с параметром имя текущего устройства вывода.

Например:

writedevice(addresses) – последующие команды записи могут использовать символическое имя файла *addresses;*

writedevice(X) – результат X= *addresses.*

По умолчанию файлы считаются текстовыми, однако установив режим работы с файлом можно работать с ним как с двоичным, однако при этом следует использовать только посимвольный ввод.

- filemode(Символическое_имя_файла, Тип Файла) (file,integer): прототип (i,i) режим: 0 – текстовый файл, 1 –двоичный файл – устанавливает тип заданного файла; прототип (i,o) – читает тип заданного файла и связывает его с параметром ″Тип файла″.

Например:

filemode(users,0) – тип файла *users* устанавливается как текстовый;

filemode(users,X) – результат X=0, если файл *users* – текстовый.

По завершению работы с файлом его следует закрыть:

- closefile(Символическое_имя_файла) (file): прототип (i) – имя файла не должно быть заключено в кавычки. Выполняется успешно, даже если файл перед этим не был открыт.

Например:

```
domains
        file = textfile
goal
openread(textfile, "goo.txt"), readdevice(textfile),
readln(Str),write(Str),closefile(textfile).
```

Файлы на диске из программы могут быть переименованы или удалены:

- renamefile(Старое_DOS_имя, Новое_DOS_имя) (string,string), (i,i)

- deletefile(Имя_файла_в_DOS) (string),(i).

К вспомогательным предикатам относятся:

- filepos(Символическое_имя_файла, Позиция_в_файле, Режим) (file,real,integer) - устанавливает указатель данного файла на заданную позицию. Режим 0 = относительно начала файла;

  1 = относительно текущего положения указателя;

  2 = относительно конца файла.

  Например:

filepos(abc,10,0) – устанавливает указатель в файле *abc* на десятом байте.

- eof(Символическое_имя_файла) (file),(i) – выполняется успешно, если указатель текущей позиции файла указывает на конец файла, и завершается неудачно в противном случае.
- flush(Символическое_имя_файла) (file), (i) – содержимое внутреннего файлового буфера пересылается в заданный файл.
- disk(DosPath) /* (string): прототип (i) – устанавливает путь и накопитель; (o) – связывает с параметром текущий накопитель и путь.

Например:

disk(″C:\Prolog″) – на накопителе C будет установлен путь \Prolog.

disk(X) – результат X = C:\Prolog\Bin\qwer.

2. Внутренняя база данных

Турбо Пролог поддерживает работу с внутренней (ВБД) и внешней (дисковой - ДБД) базами данных, а также работу с дополнительной оперативной памятью (EMS ОЗУ).

ВБД состоит из фактов, которые добавляются/удаляются и существуют в ОЗУ только во время работы программы. Однако, их можно

сохранить на диске. Для работы с ВБД Турбо Пролог использует следующие встроенные предикаты [1]:

- assert(факт) – заносит факт в базу данных перед другими фактами. В результате данный факт будет добавлен в начало базы данных. Факт должен быть термом, принадлежащим домену *dbasedom.*
- asserta(факт) - добавляет факты в ВБД перед существующими фактами.
- assertz(факт) - добавляет факты в ВБД после существующих фактов
- retract(факт) - удаление существующего факта из ВБД
- retractall(факт) - удаление существующих фактов из ВБД
- consult(имя_файла) – записывает в базу данных текстовый файл, который может быть создан в результате выполнения предиката *save*. Этот файл содержит факты, которые должны быть описаны в разделе *database*. Выполнение предиката *consult* не будет успешным, если в файле имеются синтаксические ошибки.

save(имя_файла) - сохранение ВБД в файле или на жесткий диск.

Эти предикаты могут иметь один или два аргумента. Второй аргумент является необязательным и обозначает имя ВБД. Факты в ВБД хранятся в виде таблицы, которую легко модифицировать. В ВБД можно добавлять факты, но не правила. В фактах ВБД не может быть свободных переменных. Если ВБД не дано имя, то ей будет присвоено имя dbasedom.dba.

Пример программы с использованием внутренней базы данных:

```
    domains
    database
base(integer,string,integer)
     predicates
write_base(integer)
ch_base(integer)
     clauses
 write_base(N):-base(N,Name,Group),
```

```
                write("Номер ",N),nl,
                write("Фамилия ",Name),nl,
                write("Группа ",Group),nl,
                write("----------------"),nl.
ch_base(N):-write("Фамилия:"),readln(S),
            write("Группа"),readint(S1),
            retract(base(N,_,_)),
            assert(base(N,S,S1)).
    goal
  consult("dd.dba"),
  write("Изменить запись.Номер-"),
  readint(N),
  write_base(N),
  ch_base(N),
  save("dd.dba").
```

3. Внешняя база данных

Внешняя база данных создается в случае, если объем данных больше объема свободной части ОЗУ или предполагается значительное расширение БД.

ДБД может быть расположена:

Встроенный атрибут	Местонахождение внешней БД
in_file	в файле на диске
in_memory	в оперативной памяти
in_ems	в EMS-памяти (расширение ОЗУ до 4 Мб.)

ДБД состоит из двух компонент: элементов данных (термы Турбо Пролога) и цепочек (chain), в которых хранятся термы. В цепочке может храниться неограниченное количество термов. ВДБД может быть любое число цепочек. Цепочка выбирается по имени. Имя цепочки – это просто строка символов. Для быстрого поиска данных цепочка может быть индексирована методом B+дерева [1].

Имена предикатов, работающих с ДБД, построены следующим образом: db_ <тип объекта данных><операция>.

Например: db_term_delete.

Если имя объекта не указано подразумевается вся база данных.
Объявление имени ДБД:

domains

db_selector = имя_базы1, имя_базы2, ...

Ссылка на ДБД осуществляется по ее имени.

3. 1. Предикаты для работы со всей ДБД.

db_create(имя_базы, имя_файла, расположение), (i,i,i) – создание новой ДБД:

Например:

db_create(db_name, "F.TXT", in_file)

db_create(db_mark, "marks", in_memory)

db_open(имя_базы, имя_файла, расположение), (i,i,i) – открытие ранее созданной базы данных.

db_copy(Имя_базы, имя_файла, Новое_расположение), (i,i,i) – перемещение бызы данных в другое место (например, из файла в ОЗУ или наоборот.

db_close(Имя_базы), (i) – в конце работы ДБД должна быть закрыта.

db_delete(Имя_базы), (i) – ДБД можно удалить.

3.2. Предикаты для работы с цепочками

Цепочки [3] ДБД аналогичны внутренней базе данных. Цепочки, в некотором смысле, можно отожествить с записями в обычной реляционной СУБД. Они хранят данные в форме термов Пролога (т.е. в виде сложных объектов: списков, структур; или значений простых типов данных). ДБД может хранить любые допустимые в Турбо – Прологе типы термов.

Цепочки термов запоминаются последовательно и одновременно доступна только одна цепочка. Каждая цепочка имеет собственное имя. Имя цепочки нигде специально не объявляется, цепочка создается, когда ее имя впервые встретилось в предикатах записи фактов в ДБД. Поэтому в

именах цепочек не следует допускать опечаток, иначе можно создать ненужную цепочку и таким образом "потерять данные".

Следующие предиката служат для работы с цепочками в ДБД:

chain_inserta(X,W,S,T,_) – вставляет терм в начало базы;

chain_insertz(X,W,S,T,_) – вставляет терм в конец базы;

db_chains(X,C) – создает ссылку на цепь;

chain_first(X,C,R) – позиционирует указатель на первый элемент цепи;

ref_term(X,S,R,TERM) – возвращает терм на который указывает указатель;

chain_next(X,R,NEXT) – возвращает указатель на следующий терм,

где X – селектор базы описывается в DOMAINS как DB_SELECTOR = mydba

Y – имя базы по стандарту ДОС в кавычках

Z – место расположения базы (в памяти или в виде файла)

W – описатель цепи

S – описатель структуры файла базы данных

T – терм

_ – необъязательная переменная возвращающая код ошибки если она есть

C – ссылка на описатель цепи

R – положение указателя в цепи

TERM – переменная конкретизированная термом на котором находится указатель

NEXT – переменная конкретизированная указателем на следующий терм

dbman – описатель структуры файла базы данных

4. Индексация цепочек методом B+дерева.

При добавлении термов с использованием для индексации B+дерева с каждым ссылочным числом связан ключ. Так как это число ссылается на уникальную запись (вход) БД, нахождение правильного ключа быстро

приводит к искомой записи данных. B+дерево (индексная цепочка) создается с помощью предиката:

bt_create(Имя_ДБД, Имя_B+дерева, Селектор_B+дерева, Длина_Ключа, Длина_Узла),

так как информация B+дерева хранится в том же файле ДБД, что и индексируемая цепочка, то необходимо указывать "Имя_ДБД" базы, в которой находится эта цепочка. "Имя_B+дерева" - это имя, которое ему дает пользователь; оно является строкой. Третий аргумент является выходным, он относится к специальному встроенному простому типу bt_selector и используется в ряде других предикатов для идентификации созданного B+дерева. "Длина ключа" - длина самого длинного ключа по которому осуществляется поиск записи. Однако не следует злоупотреблять этим аргументом, так как хранение слишком длинных ключей в большой ДБД может потребовать значительного объема дисковой памяти.

B+дерево разбито на отдельные страницы или узлы. Каждый узел B+дерева является либо последним, либо порождает два нижележащих узла. Каждый узел содержит группу ключей из заданного диапазона. Так как никакие два узла не содержат одинаковые значения ключей, то можно быстро проверить не находится ли искомый ключ внутри диапазона конкретного узла. Если да, то быстро находится ссылочное число, если ключ меньше, то поиск ведется по левой ветви B+дерева, если больше - по правой. Аргумент "Длина_Узла" в bt_create определяет сколько ключей запоминается в каждом узле B+дерева. Для баз данных среднего размера рекомендуется "Длина_Узла" = 4.

Как и цепочки ДБД, B+деревья могут быть закрыты и снова открыты с помощью предикатов: bt_open() и bt_close().

Чтобы B+дерево было правильно сохранено, оно должно закрываться до закрытия соответствующего файла базы данных. Как и с ДБД B+дерево может быть модифицировано удалением или добавлением ключей. При

индексировании ключи автоматически сохраняются системой. Для модификации используются два предиката: key_insert() и key_delete()

Для поиска по заданному ссылочному номеру следует использовать предикат key_search(), если заданный ключ не найден, возникает ситуация fail. Если существует 2 одинаковых ключа, то возвращается первый найденный. Чтобы найти следующий (предыдущий) нужно применить, соответственно, предикат
key_next() или key_prev(). Полный список предикатов для работы с внитренней и внешней базами данных можно получить в процессе работы нажав клавишу F1. Они находятся в файле PROLOG.HLP.

Примеры программ для работы с ДБД:

1)
```
domains
    db_selector = mmm
predicates
    count_dba(integer)
    count(REF,INTEGER,INTEGER)
clauses
    count_dba(N):- chain_first(mmm,name,REF),count(REF,1,N).
    count(REF,N,N):-ref_term(mmm,string,REF,S),write(S).
    count(REF,N,N2):-chain_next(mmm,REF,NEXT),!,N1=N+1,
    count(NEXT,N1,N2).
goal
    db_create(mmm, "aaaa.ile", in_file),
    db_statistics(mmm,_,_,_,_),
    write("\nБаза имен.\n"),
    write("Введите номер не больше 8"),
    readint(N),
    chain_inserta(mmm,name,string,"ДМИТРИЙ",_),
    chain_insertz(mmm,name,string,"АЛЕКСЕЙ",_),
    chain_insertz(mmm,name,string,"МАКСИМ",_),
```

```
        count_dba(N),
        db_close(mmm).
```

2)
```
domains
        db_selector = mydba
        dbman = person(firstname,age,city)
        firstname,city = srting
        age = integer
    predicates
        rd(ref)
    clauses
        rd(REF):-ref_term(mydba,dbman,REF,TERM),write(TERM),nl,fail.
        rd(REF):-chain_next(mydba,REF,NEXT),!,rd(NEXT).
        rd(_).
    goal
        clearwindow,
        db_create(mydba,"dd.bin",in_file),
        db_close(mydba),
        db_open(mydba,"dd.bin",in_file),
chain_inserta(mydba,mychain,dbman,person
                ("Унру Артур Яковлевич",21,"Комсомольск-на-Амуре"),_),
chain_insertz(mydba,mychain,dbman,person
                ("Смирнов Николай Семенович",20,"Благовещенск"),_),
        db_chains(mydba,CHAIN),
        chain_first(mydba,CHAIN,REF),
        rd(REF),readchar(_),
        db_delete("dd.bin",in_file).
```

Последовательность действий

1. Изучить пояснения к работе.

2. Загрузить Турбо Пролог.
3. Войти в режим редактирования (Alt-E) и ввести одну из предложенных в примере программ.
4. Включить режим трассировки и просмотреть выполнение предикатов по шагам (см. п. 2.3 предыдущей работы).

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ № 1

Тема: Работа с базами данных.

1. Разработать программу с использованием файлов, ВБД, ДБД по темам:

1.1. Самолеты: наименование типа, фамилия конструктора, год выпуска, количество кресел, грузоподъемность (т).

1.2. Расчет движения: наименование воздушной линии, тип самолета, количество рейсов, налет (тыс. км.), пассажирооборот.

1.3. Перевозки: тип самолета, номер борта, количество рейсов, налет в часах, налет (тыс. км.).

1.4. Расписание: номер рейса, наименование рейса, тип самолета, стоимость билета, протяженность линии.

1.5. Сооружения аэропорта: наименование, площадь, этажность, год сооружения, стоимость (млн. руб.).

1.6. Ремонт аэродромных сооружений: наименование, шифр, вид ремонта, стоимость ремонта, наименование подрядчика.

1.7. Кассы авиабилетов: номер кассы, Ф.И.О. кассира, количество проданных билетов, суммарная выручка, дата продажи.

1.8. Характеристики персональных компьютеров: тип процессора, тактовая частота, емкость ОП (Мбайт), емкость ЖМД (Мбайт), тип монитора.

1.9. Города: наименование, количество жителей, площадь (кв. км.), год основания, количество ВУЗов).

1.10. Московские мосты: наименование, высота, ширина, количество опор, протяженность.

1.11. Линии московского метро: наименование, район линии, год пуска, протяженность (км.), количество поездов.

1.12. Легковые автомобили: марка, цвет, стоимость, изготовитель, максимальная скорость.

1.13. Продажа программных продуктов: наименование, фирма – изготовитель, стоимость (тыс. руб.), объем (Мбайт), количество на складе.

1.14. Абонентская плата за телефон: Ф.И.О. абонента, телефон, год установки, количество абонентов, плата за телефон.

1.15. Детские сады: наименование детского сада, номер сада, количество детей, район города, плата за месяц.

1.16. Сотрудники: Ф.И.О., табельный номер, дата рождения, оклад (тыс. руб.), стаж работы.

1.17. Ведомость зарплаты за текущий месяц: Ф.И.О., номер отдела, Табельный номер, количество рабочих часов, размер зарплаты.

1.18. Музеи: наименование, назначение, адрес, время работы, стоимость билета.

1.19. Экскурсии: наименование, страна, стоимость, продолжительность, транспорт.

1.20. Кинофильмы: наименование кинотеатра, стоимость билета, время сеансов, адрес мест, количество.

1.21. Книга – почтой: наименование, Ф.И.О. автора, номер по каталогу, издательство, стоимость книги.

1.22. Квартиры: адрес, площадь (кв. м.), сторона света, стоимость (1 кв. м.), этаж, количество комнат.

1.23. Склад товаров: номер магазина, наименование товара, артикул товара, цена единицы товара, количество товара.

1.24. Телевизоры на складе магазина: наименование, фирма – изготовитель, стоимость, размер экрана, количество на складе.

1.25. Холодильники на складе магазина: наименование, фирма – изготовитель, стоимость, емкость камеры, количество на складе.

2. Пользуясь средствами Турбо Пролога добавить предикаты для поиска фактов в построенной внутренней базе данных.

3.Каждый новый введенный вами предикат проверить задавая в окне диалога после подсказки Goal: внешнюю цель с этим предикатом.

ЛАБОРАТОРНАЯ РАБОТА № 3.

УНИВЕРСАЛЬНЫЙ ГРАФИЧЕСКИЙ ИНТЕРФЕЙС В ЯЗЫКЕ ТУРБО ПРОЛОГ.

Цель: Освоение основных режимов работы с универсальным графическим интерфейсом (УГИ): создание окон, очистка окон, переход из одного окна в другое, изменение цвета фона и изображения, редактирования текста в них, удаления окон, а также построение графических объектов с помощью предикатов УГИ.

Общие сведения

1. Режимы работы монитора

1.1. Определения

Текстовый режим [4] определяет вывод на экран, который разделен на ячейки (обычно 80 колонок на 25 строк), и может отображать только символы из кодового набора ПЭВМ. Каждая ячейка содержит атрибут и символ. Атрибут говорит о том, как выводится на экран символ (его цвет, мигание, интенсивность свечения).

Текущая позиция на экране отмечается мерцающим прямоугольником - курсором. Позиция курсора X,Y (координаты точки) задаются номером колонки и номером строки символа или точки в зависимости от режима. Началом координат является левый верхний угол экрана (1,1), при этом X увеличивается слева направо, а Y – сверху вниз.

Окно – прямоугольная область экрана (или весь экран), возможно окруженная рамкой и выделенная другим цветом или оттенком. Операции над окном производятся как над целым экраном. На экране одновременно могут находиться несколько окон, которые могут перекрывать друг друга. Окно, в котором находится курсор, называется текущим.

Для кодирования цвета окон в предикатах Турбо – Пролога приняты следующие соглашения [2]:

Цвет фона	Код	Цвет изображения	Код
черный	0	черный	0
голубой	16	голубой	1
зеленый	32	зеленый	2
сиреневый	48	сиреневый	3
красный	64	красный	4
фиолетовый	80	фиолетовый	5
коричневый	96	коричневый	6
белый	112	белый	7

Для получения яркого цвета к цвету изображения нужно добавить 8. Для получения атрибута окна нужно сложить цвет фона и изображения. Например, голубое изображение на белом фоне имеет атрибут: 112+1=113.

Графический режим [4] определяет построение изображения на экране из точек (пикселов). Размер экрана измеряется количеством точек по горизонтали (X) и по вертикали (Y) и зависит от типа видеоадаптера. Число точек по осям X,Y называется разрешением экрана. Каждая точка может быть включена или выключена, а на цветных мониторах иметь еще и цвет. Таким образом изображение на экране строится включением и окраской точек.

В графическом режиме нет курсора. Вместо него используется указатель текущей позиции экрана CP, который может быть перемещен в любое место экрана, где должен строиться элемент изображения.

При наличии видеоадаптера цветной графики (CGA, MCGA, VGA) можно установить цвет фона (экрана) и цвет изображения. Цвет фона и цвет изображения являются атрибутом окна и изображения. В процессе работы атрибут можно менять.

Турбо Пролог 2.0 поддерживает следующие видеоадаптеры [1]:

Название видеоадаптера	Драйвер
CGA - Color Graphics Adapter	CGA.BGI
EGA - Enhanced Graphics Adapter	EGAVGA.BGI
MCGA - Multi Color Graphics Adapter	
HGA - Hercules Graphics Adapter	HERC.BGI
VGA - Video Graphics Array	EGAVGA.BGI
IBM8514 - Super VGA	IBM8514.BGI
AT&T - 400-строковый, ПЭВМ Оливетти	ATT.BGI

Основными видеоадаптерами являются: CGA, EGA и VGA.

Графика CGA.

Графический адаптер CGA был первым цветным графическим адаптером на ПЭВМ фирмы IBM. CGA поддерживает один текстовый режим (25*80) и два отдельных графических режима: с высоким (640*200) инизким (320*200) разрешением экрана. При низком разрешении палитра (набор цветов) одновременно выводимых на экран цветов состоит из 4 цветов, один из которых является цветом фона.

В режиме с высоким разрешением можно использовать только один цвет изображения на черном фоне.

Графика EGA.

EGA значительно расширяет графические возможности ПЭВМ. Как и CGA, видеоадаптер EGA может быть инициализирован для работы в двух режимах: низкого (640*200) и высокого (640*350) разрешения экрана. Оба эти режима позволяют выводить на экран одновременно палитру из 16 различных цветов.

Главным различием между этими режимами является количество предоставляемых буферных страниц. В режиме низкого разрешения в памяти платы EGA может быть сохранено четыре полных страницы графического экрана. Это позволяет пользовательской программе создавать графическое изображение в трех различных экранах во время вывода на экран четвертого. Эти три дополнительных экрана дают возможность строить изображения на невидимых экранах, затем быстро переходить от одного

графического экрана к другому не ожидая, пока будет нарисовано новое изображение.

В режиме высокого разрешения EGA доступны две страницы: одна не-
видимая и одна выведенная на экран.

Графика VGA.

Имеется совместимость с EGA и CGA. Максимальная палитра 256 цветов. Разрешение 640*480. Окна в графическом режиме. Как и окна текстового режима, окно в графическом режиме, именуемое в дальнейшем VP (viewport), используются для выбора конкретной части экрана для построения в нем изображения. По умолчанию VP после инициализации занимает весь экран. В отличие от окон текстового режима VP не буферизируются. Это означает, что при перекрытии окон, информация в первом из них будет потеряна.

При записи предикатов в общем виде в строке комментарий указывается тип аргумента и вид передачи аргументов (текущий шаблон), т.е. какие аргументы являются входными (i), а какие выходными (o). Некоторые предикаты могут иметь несколько различных текущих шаблонов в зависимости от того, получают они связанные или несвязанные переменные.

1.2. Работа с окнами

Основным предикатом для создания окон является предикат для определения окна:

makewindow(Ном_Окна, Атр_экрана, Атр_рамки, Заголовок, Строка, Столбец, Высота, Ширина,), где

Ном_Окна – логический номер, присваиваемый окну для ссылок в других предикатах. Номера 80-85 использовать не рекомендуется, так как они зарезервированы за пакетом TOOLBOX (см. раздел 3.);

Атр_экрана – атрибут цвета для внутренней области окна;

Атр_рамки – атрибут цвета для внешней области окна (рамки), 0 – если рамка отсутствует;

Заголовок – текст, помещаемый на верхней рамке окна;

Строка – номер строки верхнего левого угла окна;

Столбец – номер колонки верхнего левого угла окна;

Высота – высота окна в строках;

Ширина – ширина окна в колонках.

Например:

makewindow(1,7,135,"Пример окна",5,15,6,10)

Параметры должны соответствовать характеристикам аппаратуры, иначе выдается сообщение об ошибке. Созданное окно становится текущим окном. Экран может быть разбит на несколько окон.

Для перехода их одного окна в другое служит предикат

shiftwindow(Ном_Окна): прототип (i) – активизирует окно с заданным номером; прототип (o) – связывает с параметром номер текущего окна.

Например:

shiftwindow(3) – активизируется окно с номером 3;

shiftwindow(X) – результат: X=1, если текущее окно имеет номер 1.

При переходе в другое окно координаты являются локальными т.е. отсчет идет от левого верхнего угла, которое имеет координаты (0,0).

clearwindow – стирает содержимое текущего окна (очищает окно), при этом курсор перемещается в позицию (0,0);

removewindow – удаляет текущее окно, если окна не существует, выдается ошибка времени выполнения.

Управление курсором осуществляется с помощью предиката

cursor(Строка, Столбец): прототип (i,i) – передвигает курсор в текущем окне в позицию, заданную номерами строки и столбца. Номер строки может находится в диапазоне от 0 до 24, номер столбца – от 0 до 79. координаты

левого верхнего угла экрана – (0,0); прототип (o,o) – связывает номера строки и столбца, определяющие позицию курсора, с соответствующими параметрами.

Например:

cursor(5,10) – курсор устанавливается на пятой строке в десятом столбце;

cursor(Z,S) – результат: Z=12, S=40, если курсор установлен на строке 12 в столбце 40.

existwindow(Ном_Окна) – проверяет наличие окна с заданным логическим номером;

setcolor(Окно_или_рамка) – позволяет пользователю интерактивно изменять цвет окна (0) или рамки (1). При этом появляется меню, аналогичное подменю Setup/Colors в интегрированной оболочке Турбо Пролога.

attribute(Атрибут): прототип (i) – устанавливает значение атрибута, определяющее цвет фона текущего окна. Значение атрибута определяется параметром (см. табл.); прототип (o) – опрашивает атрибут текущего окна и связвает его с параметром.

Например:

attribute(88) – устанавливает розовый цвет текущего окна;

attribute(X) – если фон окна голубой, то значением переменной X будет число 24.

scrol (Число_строк, Число_столбцов): прототип (i,i) – сдвигает содержимое текущего окна на заданное число строк и столбцов. Положительные значения аргументов задают скроллинг вниз и вправо, а отрицательные - вверх и влево.

Пример программы для работы с окнами:

```
predicates
        nondeterm repeat
```

```
goal
    makewindow(1, 1, 7, "one", 5, 0, 10, 20), write("ONE"),
    makewindow(8, 8, 7, "eight", 1, 10, 10, 20), write("EIGHT"),
    makewindow(9, 9, 7, "nine", 15, 20, 10, 20), write("NINE"),
    repeat,
    random(9,X), N=X+1, shiftwindow(9),
     shiftwindow(1),
     keypressed.
clauses
     repeat.
     repeat :- repeat.
```

2. Универсальный графический интерфейс (УГИ)

УГИ – это свыше 70 встроенных предикатов для работы с графикой. Эти предикаты позволяют работать как с отдельной точкой, так и с объектами типа линий, дуг, окружностей, многоугольников и др.

УГИ поддерживает различные виды штриховки линий, закраску поверхностей, различные виды шрифтов (фонты) и драйверы практически всех распространенных графических адаптеров ПЭВМ.

Полный список предикатов УГИ можно получить в процессе работы нажав клавишу F1. Они находятся в текстовом файле PROLOG.HLP. Чтобы применять УГИ необходимо иметь видеографический адаптер, поддерживаемый УГИ (например, нельзя применять на ЕС 1840).

2.1. Установка графического режима

Режим работы графического экрана устанавливается предикатом initgraph, который инициализирует графическую систему, загружая с диска соответствующий драйвер графического видеоустройства (см. файлы с расширением BGI) переводя его в графический режим работы. Предикат initgraph() имеет пять аргументов. Он связывает переменные NewDriver и NewMode с реальным драйвером и режимом работы, а также устанавливает

значения по умолчанию всем графическим переменным (текущая позиция, цвет, палитра и т.д. в зависимости от видеоадаптера).

initgraph(Graphdriver,Graphmode,NewDriver,NewMode,Pathtodriver)

(integer,integer,integer,integer,string): прототип (i,i,o,o,i), где

Graphdriver – целое число, которое задает номер используемого драйвера, 0 означает, что система должна сама определить тип используемого в ПЭВМ видеоконтроллера и вернуть номер соответствующего ему драйвера (1 – CGA, 2 – MCGA, 3 – EGA, ..., 9 – VGA). См. также декларации в файле GRAPDECL.PRO;

Graphmode – графический режим. Символические константы для объявления режима определены в файле GRAPDECL.PRO.

Pathtodriver – путь к директории, где находится драйверы (*.BGI), пустая строка " " означает текущую директорию.

Например:

```
constants
  bgi_path = " "
goal
  InitGraph(G_Driver,G_Mode, _, _, bgi_Path)
```

После инициализации экрана он очищается и CP устанавливается равным координатам верхнего левого угла экрана (0,0). Иногда, если предполагается, что программа будет выполняться на различных ПЭВМ, более удобно перед выполнением предиката initgraph() выполнить предикат:

detectgraph(Graphdriver,Graphmode) (integer,integer): прототип (o,o) – проверяет какой графический адаптер используется в системе и определяет режим его работы при наибольшем разрешении. Если графический адаптер не обнаружен, то возвращается – 2.

Например:

DetectGraph(G_Driver, G_Mode1),

Чтобы освободить память, занятую под графику, и вернуть экран в режим, используемый до initgraph, в конце работы нужно выполнить предикат: closegraph().

2.2. Перемещение по экрану

Так как характеристики видеотерминалов значительно отличаются, то с помощью следующих предикатов рекомендуется получить максимальные значения X и Y: getmaxX(X) и getmaxY(Y).

Для перемещения по плоскости изображения используются предикаты:

getx(X) – получить текущую координату X;

gety(Y) – получить текущую координату Y;

moveto(X,Y) – переместить текущий указатель в точку X,Y;

moverel(DeltaX,DeltaY) – переместить текущий указатель на DeltaX пикселов по горизонтали и на DeltaY пикселов по вертикали.

Предикат cleardevice не имеет аргументов. Он очищает графический экран перемещает указатель текущей позиции в точку (0,0).

2.3. Установка/изменение текущих значений

Предикаты установки значений:

setlinestyle((Вид_Линии, Шаблон, Толщина) (integer,integer,integer): прототип (i,i,i) – устанавливает вид линии по умолчанию, используемый другими предикатами УГИ. Параметры: "Вид_Линии" – сплошная (0), из точек (1), центрированная (2), пунктирная(3), задаваемая пользователем (4); "Шаблон" – 16-битовый шаблон задающий вид линии (только для "Вида_Линии"=4, иначе шаблон игнорируется): сплошная линия $FFFF, пунктирная $3333; "Толщина" – нормальная линия или утолщенная.

setfillpattern(Список_шаблона, Цвет) (bgi_list,integer): прототип (i,i) – устанавливает определяемый пользователем шаблон – заполнитель. Шаблон в виде матрицы 8 на 8 бит кодируется в виде списка из 8

однобайтовых элементов, каждый из которых кодирует 8 бит. Если бит шаблона равен 1, то соответствующий ему пиксел будет выведен на экран.

setfillstyle(Шаблон, Индекс_Цвета) (integer,integer): прототип (i,i) – устанавливает в качестве текущего заполнителя один из определенных в системе шаблонов, а текущий цвет заполнителя равным "Индекс_Цвета" в палитре. Константы для "Шаблона" приведены в файле GRAPDECL.PRO

setpalette(Индекс, Реальный_Цвет) (integer,integer): прототип (i,i) – изменяет один цвет палитры, расположенный в ней под номером "Индекс", на цвет заданный параметром "Реальный_Цвет". Последний представляет собой зависящий от аппаратуры номер цвета для используемого драйвера. Индекс должен находится в диапазоне от 0 до <количество_цветов_в_текущей_палитре>. Для адаптера CGA можно изменять цвет только с индексом 0 (цвет фона). Файл GRAPDECL.PRO содержит определения констант для наиболее часто применяемых цветов.

setallpalette(Список_Цветов) (bgi_ilist): прототип (i) – изменяет все цвета палитры в соответствии со "Списком_Цветов" (адаптеры EGA/VGA).

Текущий цвет фона и цвет изображения (из текущей палитры) можно получить/установить с помощью предикатов:

getbkcolor(BkColor) (integer): прототип (o) – возвращает цвет фона;

getcolor(Color) (integer): прототип (o) – возвращает цвет изображения;

setbkcolor(Color) (integer): прототип (i) – установить цвет фона;

setcolor(Color) (integer): прототип (i) – установить цвет изображения.

Следующие два предиката позволяют получить/изменить цвет заданной точки:

getpixel(X,Y,Цвет) (integer,integer,integer): прототип (i,i,o);

putpixel(X,Y,Цвет_Точки) (integer,integer,integer): прототип (i,i,i);

где (X,Y) - координаты точки (пиксела).

2.4. Построение графических объектов

В следующих предикатах углы отсчитываются против часовой стрелки, 0 градусов соответствует трем часам.

arc(X,Y,StartAngle,EndAngle,R) (integer,integer,integer,integer,integer): прототип (i,i,i,i,i) – рисует текущим цветом дугу окружности с центром (X,Y) и радиусом R, начинающуюся с ″StartAngl″ и заканчивающуюся на ″ EndAngl″;

ellipse(X, Y, StAngle, EndAngle, Xradius, YRadius) (все аргументы целочисленные): прототип (i,i,i,i,i,i) – рисует эллипс;

pieslice(X,Y,Нач_Угол, Кон_Угол, Радиус) (все аргументы целочисленные): прототип (i,i,i,i,i) – рисует и заполняет сектор круга от ″ Нач_Угол″ до ″Кон_Угол″ с центром в (X,Y) и заданным радиусом. Граница сектора выделяется текущим цветом изображения, а заполнение производится текущим шаблоном и цветом заполнителя;

pieslicexy(X,Y,Нач_Угол, Кон_Угол, X_Радиус, Y_Радиус) (все аргументы целочисленные): прототип (i,i,i,i,i,i) – рисует и заполняет сектор эллипса с центром (X,Y), горизонтальным радиусом ″X_Радиус″, вертикальным – ″ Y_Радиус″, от ″Нач_Угол ″до ″Кон_Угол″. Остальное как для pieslice;

line(X0,Y0,X1,Y1) (все аргументы целочисленные): прототип (i,i,i,i) – рисует текущим цветом, стилем линии и толщиной прямую линию между двумя заданными точками: (X0,Y0) – начало; (X1,Y1) – конец. Предикат line не обновляет текущую позицию и всегда вычисляется;

linerel(DeltaX,DeltaY) (integer,integer): прототип (i,i) – рисует линию от текущей позиции до точки приращения координат до которой равны (DeltaX, DeltaY). Текушая позиция изменяется на (DeltaX, DeltaY);

lineto(X,Y) (integer,integer): прототип (i,i) – рисует линию от текущей позиции до точки с координатами (X,Y), после чего текущая позиция перемещается в (X,Y);

rectangle(X, Y, X1, Y1) (все аргументы целочисленные): прототип (i,i,i,i) –рисует текущим цветом и стилем линии прямоугольник по координатам его верхнего левого (X,Y) и нижнего правого (X1,Y1) углов;

drawpoly(PolyPointsList) (point_list): прототип (i) – рисует многоугольник,

где (point_list) – список координат вершин X,Y.

Например: drawpoly([45,15, 85,45, 45,85, 15,45]).

circle(X,Y,Radius) (integer,integer,integer): прототип (i,i,i) – рисует окружность.

При построении окружности учитывается отношение масштабов по горизонтали и по вертикали.

getaspectratio(Xasp,Yasp) (integer,integer): прототип (i,i) – вычисляет коэффициенты искажения по горизонтали Xasp и по вертикали Yasp; прототип (o,o) – устанавливает новые коэффициенты;

bar(Left,Top,Right,Bottom) (все аргументы целочисленные): прототип (i,i,i,i) – рисует заполненную текущим шаблоном прямоугольную полосу;

bar3d(Left,Top,Right,Bottom,Depth,Topflag) (все аргументы целочисленные): прототип (i,i,i,i,i,i) – рисует заполненную текущим шаблоном трехмерную прямоугольную полосу;

outtext(СтрокаТекста) (string): прототип (i) – выводит строку текста в окно вывода (viewport) используя текущий шрифт, размер, направление и установки выравнивания;

outtextxy(X,Y, СтрокаТекста) (integer, integer, string): прототип (i,i,i) – выводит с заданной позиции (X,Y) строку текста в окно вывода используя текущий шрифт, размер, направление и установки выравнивания;

getimage(Left,Top,Right,Bottom,BitMap) (все аргументы целочисленные): прототип (i,i,i,i,o) – сохраняет в памяти прямоугольную область экрана в переменной BitMap, которой потом уже нельзя манипулировать как другими переменными – она используется в предикате putimage для вывода сохраненного изображения на экран, при этом координаты X,Y задают

местоположение верхнего левого края области. Количество байтов требуемое для сохранения изображения может быть получено с помощью предиката imagesize.

imagesize(Левый, Верхний, Правый, Нижний, Size) (integer,integer,integer,integer,string): прототип (i,i,i,i,o) возвращает в Size колическво байтов, необходимых getimage для сохранения изображения. Сохраняемое изображение – это прямоугольник, задаваемый координатами "Левого Верхнего" и "Правого Нижнего" углов. Если для сохранения изображения требуется память больше 64К, то возвращается $FFFF.

putimage(X,Y,Bitmap, BitOp) (integer,integer,string,integer): прототип (i,i,i,i) помещает изображение, сохраненное с помощью getimage, обратно на экран, при этом (X,Y) являются координатами верхнего левого угла изображения. Аргумент BitOp задает каким образом по цвету существующего пиксела и цвету пиксела выводимого изображения вычисляется цвет каждой результирующей точки на экране:

0 - копируются цвет изображения

1 - выполняется логическое OR (исключающее ИЛИ)

2 - выполняется логическое OR (ИЛИ)

3 - выполняется логическое AND (И)

4 - копируется инверсное изображение.

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ № 1

Последовательность действий

1. Изучить пояснения к работе.

2. Войти в режим редактирования (Alt-E) и ввести определения предикатов для создания на экране трех непересекающихся окон (см. предикат makewindow).

3. Пользуясь средствами Турбо Пролога добавьте предикаты для

- перехода из одного окна в другое;
- очистки окна;
- редактирования текста в текущем окне;

- скроллинга текста в окне;
- удаления окна;
- изменения размеров окна;
- изменения цвета окна и рамки;

4. После ввода каждого нового предиката, программу следует откомпилировать для чего нажать клавиши ALT-F9. Если при компиляции будут обнаружены синтаксические ошибки, то их следует тут же исправить и добиться безошибочной компиляции (см. п. 2.2).

5. Если компиляция прошла успешно, перейти в режим исполнения: Run в главном меню или нажатие клавиш Alt-R.

6. Каждый новый введенный Вами предикат проверьте задавая в окне диалога после подсказки Goal: внешнюю цель с этим предикатом.

7. Включить режим трассировки и просмотреть выполнение предиката по шагам (см. п. 2.3).

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ № 2

1. Инициализировать работу с графикой (предикат initgraph).

2. Построить графические объекты согласно номера варианта.

2.1. Программа строит на экране два равносторонних треугольника и окрашивает треугольники, а также пространство между ними в разные цвета.

2.2. На экране строится окружность, затем внутри нее строятся три радиуса под углом 120 градусов, составляющие трехлучевую звезду, которая совершает поворот вокруг своего центра против часовой стрелки на 240 градусов.

2.3. На экране строится окружность, затем внутри нее строится радиус, который совершает один оборот против часовой стрелки.

2. - построить простые графические объекты: круги, дуги,

VII МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ВЫПОЛНЕНИЮ ДОМАШНИХ ЗАДАНИЙ И КОНТРОЛЬНЫХ РАБОТ

При выполнении домашних заданий необходимо повторить материал лекций и других видов занятий. Изучить соответствующие разделы учебно-методических пособий и учебной литературы.

При подготовке к контрольной работе повторить изученный материал, выполнить задания для самостоятельной проверки и ответить на контрольные вопросы из соответствующих разделов методических пособий.

VIII МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ

Межсессионный контроль осуществляется на основе выполнения домашних заданий, контрольных работ, лабораторных заданий.

По итогам их выполнения и контроля самостоятельно изученных теоретическим вопросов в сроки, установленные деканатом (как правило, на 6-ой и 12-ой неделе семестра) преподавателем выставляется аттестационная оценка.

IX ОСНОВНЫЕ КРИТЕРИИ ОЦЕНКИ ЗНАНИЙ СТУДЕНТОВ ПО ДИСЦИПЛИНЕ

Необходимым допуском на зачет является посещение всех занятий, проявление активности в аудитории, выполнение всех лабораторных работ, а также сдача письменного отчета по самостоятельной работе. Форма сдачи зачета – устная. Знания и умения обучающегося определяются оценками «зачтено» и «незачтено». Критерии приведены в таблице.

Основные критерии оценки знаний студентов на зачете

Оценка	Полнота, системность, прочность знаний	Обобщенность знаний

«зачтено»	Изложение полученных знаний в устной, письменной или графической форме, полное, в системе, в соответствии с требованиями учебной программы; допускаются единичные несущественные ошибки, самостоятельно исправляемые студентами	Выделение существенных признаков изученного с помощью операций анализа и синтеза; выявление причинно-следственных связей; формулировка выводов и обобщений; свободное оперирование известными фактами и сведениями с использованием сведений из других предметов
	Изложение полученных знаний в устной, письменной и графической форме, полное, в системе, в соответствии с требованиями учебной программы; допускаются отдельные несущественные ошибки, исправляемые студентами после указания преподавателя на них	Выделение существенных признаков изученного с помощью операций анализа и синтеза; выявлений причинно-следственных связей; формулировка выводов и обобщений, в которых могут быть отдельные несущественные ошибки; подтверждение изученного известными фактами и сведениями
	Изложение полученных знаний неполное, однако это не препятствует усвоению последующего программного материала; допускаются отдельные существенные ошибки, исправленные с помощью преподавателя	Затруднения при выполнении существенных признаков изученного, при выявлении причинно-следственных связей и формулировке выводов
«незачтено»	Изложение учебного материала неполное, бессистемное, что препятствует усвоению последующей учебной информации; существенные ошибки, неисправляемые даже с помощью преподавателя	Бессистемное выделение случайных признаков изученного; неумение производить простейшие операции анализа и синтеза; делать обобщения, выводы

X КАРТА ОБЕСПЕЧЕННОСТИ ДИСЦИПЛИНЫ КАДРАМИ ПРОФЕССОРСКО-ПРЕПОДАВАТЕЛЬСКОГО СОСТАВА

Все виды занятий по данной дисциплине ведет ст. преподаватель, Назаренко Наталья Викторовна.

Наталья Викторовна Назаренко,
ст. преподаватель кафедры ИиУС АмГУ

Учебно-методический комплекс по дисциплине «Представление знаний в информационных системах»

Изд-во АмГУ. Подписано к печати Формат 60x84/16. Усл. печ. л.
Тираж Заказ