Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра информационных технологий автоматизированных систем

О.В. Герман, Н.В. Батин

ЭКСПЕРТНЫЕ СИСТЕМЫ

Лабораторный практикум
для студентов специальности
«Автоматизированные системы обработки информации»
дневной и дистанционной форм обучения

Библиотека БГУИР

Минск 2003

УДК 681.322 + 007 (076)
ББК 32.813 Я 73
Г 38

Р е ц е н з е н т:
профессор кафедры информатики и вычислительной техники БГТУ,
д-р техн. наук В.Л. Колесников

Герман О.В.

Г 38 Экспертные системы: Лабораторный практикум для студентов специальности "Автоматизированные системы обработки информации" дневной и дистанционной форм обучения / О.В. Герман, Н.В. Батин. - Мн.: БГУИР, 2003.- 75 с.

ISBN 985-444-473-2

В лабораторном практикуме приводится практический материал, связанный с разработкой экспертных систем. Рассматривается реализация основных механизмов экспертных систем в среде программирования Visual Prolog.

Лабораторный практикум рекомендуется студентам специальности "Автоматизированные системы обработки информации", изучающим перспективные интеллектуальные компьютерные технологии. Практикум может использоваться студентами в курсовом и дипломном проектировании.

УДК 681.322 + 007 (076)

ББК 32.813 Я 73

ISBN 985-444-473-2

© О.В. Герман, Н.В. Батин, 2003
© БГУИР, 2003

Библиотека БГУИР

СОДЕРЖАНИЕ

Лабораторная работа №1. Основные сведения о языке Пролог и системе программирования Visual Prolog

Лабораторная работа №2. Обработка списков в программах на Прологе

Лабораторная работа №3. Развитые типы данных в программах на Прологе. Базы данных в программах на Прологе

Лабораторная работа №4. Представление и обработка знаний с использованием логических функций

Лабораторная работа №5. Построение базы знаний продукционной экспертной системы

Лабораторная работа №6. Построение механизма вывода в продукционной экспертной системе

Лабораторная работа №7. Методы и алгоритмы распознавания в экспертных системах

Лабораторная работа №8. Байесовская стратегия оценки выводов

Литература

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №1

ОСНОВНЫЕ СВЕДЕНИЯ О ЯЗЫКЕ ПРОЛОГ И СИСТЕМЕ ПРОГРАММИРОВАНИЯ VISUAL PROLOG

Цель работы. Изучение основных возможностей языка Пролог и системы программирования Visual Prolog. Изучение механизмов управления в программах на языке Пролог.

1.1. Начало работы с системой Visual Prolog

Программы, разрабатываемые в системе Visual Prolog (VIP), оформляются как проекты. Они могут строиться на различных платформах (например, MS DOS или Windows). От этого зависит уровень возможностей программы. В данной работе рассматривается создание программы на платформе MS-DOS.

Начальные действия по созданию новой программы в среде VIP выполняются в следующем порядке.

1. Запустить систему VIP.
2. Из меню Project выбрать команду New Project.
3. В появившемся на экране окне Application Expert выбрать закладку General (обычно она оказывается выбранной по умолчанию). В поле Project Name указать имя проекта (например, LAB1). В поле Name of .VPR file достаточно просто щелкнуть мышью; будет указано имя файла проекта, совпадающее с заданным именем проекта (в данном примере – LAB1.VPR). В поле Base Directory указывается имя папки, в которой должны сохраняться создаваемые файлы программы. Например, если их требуется сохранять на диске E: в папке ES, то следует указать E:\ES\
4. Перейти на закладку Target. В поле Platform выбрать DOS, в поле UI Strategy – TextMode, в поле Target Type – exe, в поле Main Program – Prolog. Нажать кнопку Create.
5. На экран выводится окно, в котором отображается структура проекта. Для начала набора текста программы следует выделить имя программы (файл с расширением .PRO) и нажать кнопку Edit, или просто дважды щелкнуть мышью по имени программы.

Если проект уже имеется на диске, то для его загрузки следует воспользоваться командой Open Project из меню Project.

Библиотека БГУИР

1.2. Подготовка и запуск программы в системе Visual Prolog

После выполнения действий, описанных выше, на экране появляется “шаблон” программы. Он содержит разделы predicates, clauses и goal, имеющиеся в любой программе на языке Пролог. Назначение этих разделов будет рассмотрено ниже. Кроме того, на экран выводятся комментарии; из них можно видеть, что в Прологе начало комментария обозначается символами /*, а конец - */. Выводятся еще некоторые команды; для рассматриваемого примера их можно удалить.

В качестве примера рассмотрим подготовку и запуск программы, запрашивающей у пользователя два числа и выводящей на экран результат их умножения. Текст программы приведен ниже.

```
predicates

nondeterm obrab
nondeterm umnoz (real, real, real)

clauses

obrab:- write ("Введите 1-е число: "), readreal (X1),
         write ("Введите 2-е число: "), readreal (X2),
        umnoz (X1, X2, Y), write ("Результат: ",Y).

umnoz (A, B, C):- C=A*B.

goal
  obrab.
```

Смысл конструкций, использованных в программе, будет рассмотрен ниже. Однако уже из этого примера видно, что для вывода данных на экран в Прологе используется команда write, а для ввода *вещественных* чисел – readreal. Здесь же можно видеть, что арифметические операции (например, умножение) записываются в Прологе так же, как в других языках.

Для сохранения программы следует использовать команду File – Save, или нажать клавишу F2.

Для запуска программы на выполнение следует из меню Project выбрать команду Run. На экран сначала выводится окно сообщений (Messages), где выводится информация о ходе компиляции программы.

Затем на экран выводится окно MS-DOS. В нем сначала выводится информация о версии используемой системы VIP и сообщение Press any key. Для продолжения работы необходимо нажать любую клавишу. Затем выполняется запрос исходных данных и вывод результатов. После этого окно MS-DOS следует закрыть. Происходит возврат в окно сообщений. Чтобы снова

Библиотека БГУИР

перейти к программе, следует из меню Window выбрать окно файла программы (с расширением .PRO) или файла проекта (.VPR).

В случае ошибок в программе на экран выводится окно сообщений (Message) с текстом сообщений об ошибках. Чтобы перейти к тому месту в программе, где была обнаружена ошибка, следует щелкнуть мышью по сообщению об ошибке.

1.3. Основные конструкции языка Пролог. Понятие предиката

Пролог - язык логического программирования, предназначенный для представления и обработки знаний о некоторой предметной области. В Прологе реализован так называемый декларативный подход, при котором задача описывается с помощью набора утверждений о некоторых объектах и правил обработки этих утверждений.

Основной конструкцией языка Пролог является *предикат*. Предикат - это функция от некоторого набора аргументов; при этом аргументы могут иметь любой вид, а функция принимает значение "ложь" или "истина". В языке Пролог имеется три вида предикатов: предикаты-факты, предикаты-правила и стандартные предикаты.

Предикаты-факты. Аргументами предикатов-фактов являются константы. Предикаты-факты предназначены для записи некоторых утверждений, которые при выполнении программы считаются истинными. Пусть, например, требуется записать в программе утверждение: "Иван - отец Петра". В программе на Прологе его можно выразить фактом: father ("Иван","Петр"). Точка в конце факта обязательна. Здесь father - имя предиката (оно может быть любым другим). Этот предикат имеет два аргумента, оба - строковые.

Приведем еще несколько примеров фактов. Пусть требуется записать в программе на Прологе следующую информацию: “Сотрудник Иванов работает над проектом с шифром П20. Этот проект разрабатывается по заказу завода “Рубин”, и стоимость контракта на его разработку составляет 700 тыс ден. ед.”. В программе на Прологе это можно записать в виде двух предикатов-фактов: rabota (“Иванов”, “П20”) и proekt (“П20”, “Рубин”, 700). Предикат rabota имеет два аргумента (оба – строковые), предикат proekt – три аргумента (два строковых и один целочисленный).

Набор фактов, имеющихся в программе на Прологе, называется базой данных.

Библиотека БГУИР

Предикаты-правила. Они записываются в виде: <Предикат1> истинен, если истинны <Предикат2>, <Предикат3>,...,<ПредикатN>. Здесь <Предикат1> называется головным (или заголовком), остальные - телом правила.

Приведем пример. Пусть в программе есть факты, задающие отношение "отец" (см. пример выше), а требуется составить правило, позволяющее найти по этим данным брата конкретного человека (или проверить, являются ли два конкретных человека братьями). Это правило можно сформулировать так: "Один человек - брат другого, если эти люди разные и у них один и тот же отец". На Прологе это можно записать так:

```
brother (X, Y):-  father (Z, X), father (Z, Y), X<>Y.
```

Здесь предикат brother - заголовок правила, остальные три предиката составляют его тело. Символы :- обозначают "если", запятая - союз "И" (отметим также, что точка с запятой обозначает союз "ИЛИ").

Приведем еще один пример. Пусть в программе имеются факты rabota, описывающие работу сотрудников над проектами, и факты proekt, описывающие заказчиков проектов (см. пример выше). Требуется составить правило, позволяющее найти заказчиков, для которых работает конкретный сотрудник. Такое правило можно записать следующим образом:

```
rab_zak (Fam, Zak):- rabota (Fam, Shifr), proekt (Shifr, Zak, Stoim).
```

Одно правило может иметь несколько вариантов (альтернатив). Рассмотрим пример такого правила. Пусть требуется вычислить функцию $Y=2*X$, если $X<10$, и $Y=X/2$, если $X\geq10$. На Прологе правило для такого расчета можно записать так:

```
funct (X, Y):- X<10, Y=2*X.
funct (X, Y):- X >= 10, Y=X/2.
```

Можно записать то же самое короче:

```
funct(X, Y):- X<10, Y=2*X, !.
funct(X, Y):- Y=X/2.
```

Смысл символа "!" будет показан позже.

Правила и факты называют также клозами (предложениями). Предикат brother (см. выше) состоит из одного клоза, предикат funct - из двух. Если предикат состоит из нескольких клозов, то они должны быть расположены вместе (между ними не должно быть клозов других предикатов).

Библиотека БГУИР

Стандартные предикаты входят в состав самого языка Пролог. Простейшие из них: write (X) - вывод значения X на экран (по умолчанию) или на другое устройство, где X - переменная или константа; nl - переход в следующую строку на экране; readln (X) - ввод строковой переменной; readint (X) - ввод целочисленной переменной; readreal (X) - ввод вещественной переменной.

1.4. Пример программы на Прологе. Структура программы

Пример. Имеются факты rabota, описывающие работу сотрудников над проектами, и факты proekt, описывающие заказчиков проектов (см. пример выше). Требуется составить программу, которая будет выводить шифры и заказчиков проектов, над которыми работает указанный сотрудник, если стоимость этих проектов превышает некоторую величину. Фамилия сотрудника и минимальная стоимость проекта запрашиваются у пользователя.

```
predicates

nondeterm vyvod
nondeterm poisk (string, integer)
nondeterm rabota (string, string)
nondeterm proekt (string, string, integer)

goal

vyvod.

clauses

vyvod:- write ("Фамилия: "), readln (F),
        write ("Стоимость: "), readint (S),
        poisk (F, S).

poisk (Fam, St):- rabota (Fam, Shifr),
                  proekt (Shifr, Zak, Stoim),
                  Stoim >= St,
                  write (Shifr,"  ",Zak).

rabota ("Антонов", "П20").
rabota ("Иванов", "П70").
rabota ("Иванов", "П100").
rabota ("Петров", "П100").
rabota ("Васильев", "П70").
rabota ("Иванов", "П120").
rabota ("Васильев", "П120").

proekt ("П20", "Рубин", 700).
proekt ("П100", "Рубин", 1400).
proekt ("П70", "Горизонт", 500).
proekt ("П120", "Кристалл", 1100).
```

Библиотека БГУИР

Здесь показаны три основных раздела программы на Прологе:

• predicates - список имен предикатов и типы их аргументов;

• goal – целевой предикат (цель);

• clauses - перечень клозов, т.е. правил и фактов.

Другие разделы, которые могут присутствовать в программах на Прологе, будут рассмотрены ниже.

Еще раз следует обратить внимание, что имена переменных в программах на Прологе должны начинаться с *заглавной* буквы.

1.5. Принцип работы программ на Прологе

Выполнение программы на Прологе заключается в доказательстве целевого предиката. Этот предикат обычно является правилом. Чтобы доказать правило (любое, а не только цель), требуется доказать все предикаты, составляющие его тело, т.е. найти факты, соответствующие этим предикатам. Для этого происходит согласование предиката с другим одноименным предикатом, т.е. сопоставление (*унификация*) соответствующих аргументов этих предикатов. Согласование предикатов выполняется успешно, если успешна унификация всех аргументов предикатов. При унификации аргументов возможны следующие основные случаи:

• сопоставление двух констант - заканчивается успешно, если они равны, и неудачно, если они не равны;

• сопоставление константы и переменной, еще не имеющей значения (свободной) - заканчивается успешно, и переменная получает значение константы (становится связанной);

• сопоставление двух связанных (т.е. имеющих значения) переменных – заканчивается успешно, если значения переменных равны, и неудачно, если они не равны;

• сопоставление двух переменных, одна из которых связана, а другая свободна (т.е. еще не получила значение) – заканчивается успешно, и свободная переменная принимает то же значение, что и связанная;

• сопоставление двух свободных переменных – заканчивается успешно; если впоследствии одна из переменных получает значение, то и другой переменной присваивается то же значение.

Если согласование предикатов закончилось неудачно, то делается попытка согласования данного предиката с другим одноименным клозом. Если таких клозов нет, то происходит возврат (*бэктрекинг*) к ближайшей "развилке",

Библиотека БГУИР

т.е. к точке программы, в которой было возможно другое согласование предикатов.

Рассмотрим принцип работы программ на Прологе на приведенном примере. Выполнение этой программы состоит в доказательстве целевого предиката vyvod. Для этого требуется доказать предикат poisk (предикаты write, readln и readint, также имеющиеся в теле предиката poisk, являются стандартными, и их доказательство состоит в выполнении соответствующих операций вывода и ввода).

Пусть на запрос о фамилии введено "Иванов", а на запрос о стоимости – 1000. Таким образом, переменная F связана со значением "Иванов", а переменная S – со значением 1000. Для доказательства предиката poisk выполняется его согласование с заголовком предиката-правила poisk. Выполняется успешная унификация переменных Fam и F, St и S. Чтобы доказать предикат-правило poisk, требуется доказать предикаты, входящие в его тело. Сначала доказывается предикат rabota (Fam, Shifr), где переменная Fam связана со значением "Иванов", а переменная Shifr – пока свободна. Сопоставление этого предиката с фактом rabota ("Антонов", "П20") завершается неудачно, так как не совпадают первые аргументы (фамилии). Поэтому происходит сопоставление предиката rabota (Fam, Shifr) со следующим фактом: rabota ("Иванов", "П70"). Это сопоставление успешно, и переменная Shifr связывается со значением "П70". Таким образом, предикат rabota (Fam, Shifr) доказан.

Доказывается следующий предикат в теле правила poisk: proekt (Shifr, Zak, Stoim), где переменная Shifr связана со значением "П70", а переменные Zak и Stoim пока свободны. Сопоставление этого предиката с фактами proekt ("П20", "Рубин", 700) и proekt ("П100", "Рубин", 1400) завершается неудачей из-за несовпадения первого аргумента. Сопоставление с фактом proekt ("П70", "Горизонт", 500) завершается успешно; переменная Zak связывается со значением "Горизонт", а Stoim – со значением 500.

Доказывается предикат Stoim>St. Так как Stoim=500, а St=1000, этот предикат имеет значение "ложь". В результате происходит возврат. Ближайшая "развилка" - предикат proekt (Shifr, Zak, Stoim), так как он еще не сопоставлялся с предикатом proekt ("П120", "Кристалл", 1100). При возврате происходит освобождение переменных Zak и Stoim (т.е. они снова не связаны ни с каким значением).

Сопоставление предикатов proekt (Shifr, Zak, Stoim) и proekt ("П120", "Кристалл", 1100) завершается неудачей из-за несовпадения первых аргументов

Библиотека БГУИР

(так как Shifr=”П70”). Поэтому снова происходит возврат. Теперь ближайшая “развилка” – предикат rabota (Fam, Shifr), так как он еще не сопоставлялся с пятью фактами rabota. Переменная Shifr снова становится свободной (она освобождается при возврате), переменная Fam связана со значением “Иванов”. Сопоставление предикатов rabota (Fam, Shifr) и rabota ("Иванов", "П100") выполняется успешно, и переменная Shifr связывается со значением “П100”. Таким образом, предикат rabota (Fam, Shifr) доказан.

Снова доказывается предикат proekt (Shifr, Zak, Stoim), где переменная Shifr связана со значением “П100”, а переменные Zak и Stoim свободны. Сопоставление этого предиката с фактом proekt ("П20", "Рубин", 700) заканчивается неудачей (из-за несовпадения первого аргумента), а сопоставление с фактом proekt ("П100", "Рубин", 1400) - завершается успешно. Переменная Zak связывается со значением “Рубин”, а Stoim – со значением 1400.

Снова доказывается предикат Stoim>St. Так как Stoim=1400, а St=1000, этот предикат доказывается успешно. Поэтому выполняется следующий предикат write, выводящий на экран значения переменных Shifr (“П100”) и Zak (“Рубин”). Таким образом, доказан предикат poisk, так как доказаны все предикаты, составляющие его тело. Предикат poisk является последним в целевом предикате vyvod, поэтому предикат vyvod также доказан. Выполнение программы на этом завершается.

Можно сказать, что предикат vyvod в этой программе был основной целью, а предикаты poisk, rabota и другие - временными целями.

Следует отметить, что программа вывела не все проекты стоимостью свыше 1000 ден. ед., над которыми работает указанный сотрудник (Иванов). Из набора фактов легко видеть, что он работает также над проектом П120 стоимостью 1100 ден. ед. Однако программа вывела только проект П100, указанный первым. Устранение этого недостатка будет рассмотрено ниже.

1.6. Механизмы управления в программах на Прологе

1.6.1. Искусственный возврат (fail)

Изменим рассмотренную выше программу таким образом, чтобы на экран выводился список всех проектов, соответствующих заданным условиям. Для этого достаточно записать предикат poisk в следующем виде:

```
poisk (Fam, St):- rabota (Fam, Shifr),
                  proekt (Shifr, Zak, Stoim),
                  Stoim >= St,
                  write (Shifr,"  ",Zak), nl, fail.
```

Библиотека БГУИР

poisk (_, _).

Стандартный предикат fail вызывает искусственный возврат (состояние неудачи).

Программа с измененным предикатом poisk выполняется сначала так же, как и рассмотренная выше. После того, как на экран выводятся значения переменных Shifr (“П100”) и Zak (“Рубин”), возникает состояние искусственной неудачи (fail). Происходит возврат к ближайшей "развилке" proekt (Shifr, Zak, Stoim), так как он еще не сопоставлялся с двумя фактами proekt. При этом Shifr=”П100”, а переменные Zak и Stoim при возврате становятся свободными. Сопоставление предиката proekt (Shifr, Zak, Stoim) с фактами proekt ("П70", "Горизонт", 500) и proekt ("П120", "Кристалл", 1100) заканчивается неудачей из-за несовпадения первого аргумента.

Происходит возврат к следующей развилке: rabota (Fam, Shifr), где Fam=”Иванов”, а переменная Shifr освобождается при возврате. Предикат rabota (Fam, Shifr) еще не был сопоставлен с четырьмя последними фактами rabota. Сопоставление этого предиката с фактами rabota ("Петров", "П100") и rabota ("Васильев", "П70") заканчивается неудачей. Сопоставление с фактом rabota ("Иванов", "П120") выполняется успешно, и переменная Shifr связывается со значением "П120". Таким образом, предикат rabota (Fam, Shifr) доказан.

Снова доказывается предикат proekt (Shifr, Zak, Stoim), где переменная Shifr связана со значением “П120”, а переменные Zak и Stoim – пока свободны. Сопоставление этого предиката с фактами proekt ("П20", "Рубин", 700), proekt ("П100", "Рубин", 1400) и proekt ("П70", "Горизонт", 500) заканчивается неудачей. Сопоставление с фактом proekt ("П120", "Кристалл", 1200) выполняется успешно. Переменная Zak связывается со значением “Кристалл”, а Stoim – со значением 1100.

Снова доказывается предикат Stoim>St. Так как Stoim=1100, а St=1000, этот предикат доказывается успешно. Поэтому выполняется предикат write, выводящий на экран значения переменных Shifr (“П120”) и Zak (“Кристалл”). После этого снова возникает состояние искусственной неудачи, создаваемое стандартным предикатом fail.

Ближайшей “развилкой” теперь оказывается предикат rabota (Fam, Shifr), так как он еще не был сопоставлен с предикатом rabota ("Васильев", "П120"). При этом Fam=”Иванов”, а Shifr – свободна (освобождается при возврате). Сопоставление предикатов rabota (Fam, Shifr) и rabota ("Васильев", "П120") выполняется неудачно из-за несовпадения первого аргумента.

Библиотека БГУИР

Таким образом, доказательство первого клоза предиката poisk завершилось неудачей. Поэтому происходит возврат к следующей "развилке" - ко второму клозу предиката poisk. В нем никаких действий не требуется, поэтому он доказывается успешно. Предикат poisk - последний в целевом предикате vyvod. Поэтому целевой предикат также доказан.

Использованные во втором клозе предиката poisk символы “_” (знак подчеркивания) используются для заполнения позиции аргумента, конкретное значение которого не требуется.

Стандартный предикат nl предназначен для перехода в следующую строку при выводе данных на экран.

1.6.2. Повторение

Усовершенствуем рассмотренную выше программу таким образом, чтобы после вывода ответа на экран выводился запрос “Продолжить ?”. При положительном ответе (при вводе буквы “д”) программа должна снова запрашивать у пользователя фамилию сотрудника и стоимость проектов, а затем выводить информацию о проектах, как показано выше. При отрицательном ответе (“н”) программа должна заканчивать работу. Программа при этом будет иметь следующий вид.

```
predicates

nondeterm vyvod
nondeterm poisk (string, integer)
nondeterm rabota (string, string)
nondeterm proekt (string, string, integer)
nondeterm repeat

goal

vyvod.

clauses

vyvod:- repeat,
        clearwindow, write ("Фамилия: "), readln (F),
        write ("Стоимость: "), readint (S),
        poisk (F, S),
        nl, write (“Продолжить ? ”), readchar(Prod), Prod=’н’.

poisk (Fam, St):- rabota (Fam, Shifr),
                  proekt (Shifr, Zak, Stoim),
                  Stoim >= St,
                  write (Shifr,"  ",Zak), nl, fail.
poisk (_, _).

repeat.
repeat:-repeat.
```

Библиотека БГУИР

База данных (набор фактов) остается такой же, как в предыдущих примерах.

Многократное повторение работы программы обеспечивается с помощью предиката repeat. Этот предикат работает следующим образом. При первом обращении к предикату repeat доказывается его первый клоз (repeat.), а второй (repeat:-repeat.) остается в качестве неиспользованной альтернативы. Если на запрос "Продолжить?" ввести любой ответ, отличный от “н”, то сравнение Prod='н' заканчивается неудачей и происходит возврат к ближайшей “развилке”. В данном случае такая “развилка” – предикат repeat, так как у него имеется неиспользованный клоз (repeat:- repeat.). Происходит обращение к этому клозу. Предикат repeat вызывает сам себя и успешно доказывается в *первом* клозе, а *второй* клоз снова остается в качестве неиспользованной альтернативы. После доказательства предиката repeat повторяются запросы фамилии и стоимости, и выводится информация о проектах. Если затем на запрос "Продолжить ?" снова ввести ответ, отличный от “н”, то повторяется возврат к предикату repeat. Такое выполнение программы повторяется, пока на запрос "Продолжить ?" не будет введен ответ “н”. В этом случае сравнение Prod=’н’ завершается успешно, целевой предикат доказывается, и выполнение программы завершается.

Важно понимать, что предикат repeat не является стандартным. Поэтому его объявление в разделе predicates и описание в разделе clauses является обязательным. Использование названия repeat (по-английски – “повторение”) для организации повторения программ на Прологе стало традиционным, однако можно использовать и любое другое название этого предиката, например, povtor.

Использованный в данной программе стандартный предикат clearwindow предназначен для очистки экрана (точнее, для очистки текущего окна). Стандартный предикат readchar предназначен для ввода данных типа char (одиночные символы). Из приведенного примера также видно, что данные типа char в программе на Прологе заключаются в *одиночные* кавычки.

1.6.3. Отсечение

Как показано выше, если предикат имеет несколько клозов, то при доказательстве одного из них остальные сохраняются в памяти ЭВМ как нерассмотренные. В случае неудачи при доказательстве какого-либо предиката, происходит возврат к таким нерассмотренным клозам. Если требуется исключить такую возможность, то применяется стандартный предикат отсечения (!).

Библиотека БГУИР

Пример. Имеются факты, описывающие работу сотрудников над проектами и характеристики проектов (см. примеры выше). Требуется составить программу, которая будет запрашивать фамилию сотрудника и название заказчика; если указанный сотрудник работает над каким-либо проектом данного заказчика, то программа должна выводить ответ “да”, в противном случае – “нет”.

```
predicates

nondeterm vyvod
nondeterm otvet (string, string)
nondeterm rabota (string, string)
nondeterm proekt (string, string, integer)
nondeterm repeat

goal

vyvod.

clauses

vyvod:- repeat,
        clearwindow, write ("Сотрудник: "), readln (F),
        write ("Заказчик: "), readint (Z),
        otvet (F, Z),
        nl, write (“Продолжить ? ”), readchar(Prod), Prod=’н’.

otvet (Fam, Zak):- rabota (Fam, Shifr),
                   proekt (Shifr, Zak, _),
                   write ("Да"), nl, !.
otvet (_, _):- write ("Нет"), nl.

repeat.
repeat:-repeat.
```

Пусть, например, введена фамилия сотрудника “Иванов”, заказчик – “Горизонт”. Доказывается предикат otvet (F, Z), где F=”Иванов”, Z=”Горизонт”. Предикат rabota (Fam, Shifr) успешно сопоставляется с предикатом-фактом rabota (“Иванов”, “П70”), а предикат proekt (Shifr, Zak, _) – с предикатом-фактом proekt ("П70", "Горизонт", 500). Выполняется следующий предикат (write), и на экран выводится ответ “Да”. Предикат отсечения (!) удаляет из памяти второй клоз предиката otvet, где предусмотрен вывод ответа “Нет”. Если затем на запрос "Продолжить ? " ответить “д”, то произойдет возврат к ближайшей “развилке”. В данном случае такой “развилкой” оказывается предикат repeat. Он обеспечивает повторение выполнения программы, как показано в п.1.6.2.

Если не указать в этой программе предикат !, то ответ “Да”все равно будет выведен, но второй клоз предиката otvet сохранится в качестве

Библиотека БГУИР

неиспользованной альтернативы. Затем будет задан вопрос "Продолжить ? ". При ответе “д” произойдет возврат, и ближайшей “развилкой” окажется неиспользованный второй клоз предиката otvet. На экран будет выведено сообщение “Нет” (хотя это не требуется), и снова задан вопрос "Продолжить ? ".

Порядок выполнения работы

Разработать и отладить программу в среде Visual Prolog по заданию, выданному преподавателем. В программе должны быть использованы основные механизмы управления языка Пролог: возврат, отсечение, повторение.

Контрольные вопросы

1. Понятие предиката. Виды предикатов в Прологе.
2. Примеры предикатов-фактов, правил и стандартных предикатов.
3. Структура программы на Прологе.
4. Принцип работы программ на Прологе (на примере программы, отлаженной в ходе выполнения работы).
5. Механизмы управления в Прологе.
6. Искусственный возврат
7. Повторение. Предикат repeat.
8. Отсечение.

ЛАБОРАТОРНАЯ РАБОТА №2

ОБРАБОТКА СПИСКОВ В ПРОГРАММАХ НА ПРОЛОГЕ

Цель работы. Изучение возможностей представлений и обработки данных в программах на языке Пролог с использованием списков.

2.1. Рекурсия

Прежде чем приступать к рассмотрению обработки и применения данных типа “список”, необходимо рассмотреть операцию рекурсии, так как при обработке списков она применяется практически всегда.

Рекурсия - это вызов предиката из тела самого предиката. В лабораторной работе 1 уже был рассмотрен пример рекурсии: использование предиката repeat.

Библиотека БГУИР

Рекурсия обычно применяется при обработке списков, строк (например, для поиска и замены подстроки), при вычислениях (например, вычисление сумм, факториала) и в ряде других случаев.

Приведем типичный пример программы с использованием рекурсии: вычисление факториала. Правило для вычисления факториала можно сформулировать следующим образом: если число - 0, то его факториал равен 1, в противном случае факториал числа N - это произведение (N-1)! на N.

```
predicates

nondeterm start
nondeterm fact (integer, long)

clauses

start:- clearwindow,
        write ("Введите число: "), readint (N),
        fact (N, Nf), write ("Факториал: ", Nf).

fact (0, 1):- !.
fact (N, Nf):- N1=N-1, fact (N1, N1f), Nf=N1f*N.

goal

  start.
```

Рассмотрим работу программы при вычислении 2!. При вызове предиката fact сначала рассматривается его первый клоз. Однако попытка сопоставления предикатов fact (N, Nf) и fact (0, 1) заканчивается неудачей из-за несовпадения первого аргумента, так как N=2. Рассматривается второй клоз предиката fact. Вычисляется переменная N1=2-1=1, и вызывается предикат fact (N1, N1f), где N1=1 (переменная N1f пока свободна). Таким образом, выполняется рекурсия (предикат fact вызывает сам себя). Рассматривается первый клоз предиката fact, т.е. делается попытка сопоставления предикатов fact (N1, N1f) и fact (0, 1). Сопоставление заканчивается неудачей, так как N1=1. Рассматривается второй клоз предиката fact: происходит успешное сопоставление предикатов fact (N1, N1f) и fact (N, Nf). В результате этого переменная N получает значение 1. Выполняется также унификация переменных N1f и Nf (пока они обе свободны, но когда одна из них получит значение, другой будет присвоено то же значение). Важно понимать, что при этом в памяти *сохраняется* информация о том, что обработка *предыдущего* вызова предиката fact (с аргументом N=2) *еще не закончена*. Вычисляется переменная N1=N-1=1-1=0, и вызывается предикат fact (N1, N1f). Для его доказательства рассматривается первый клоз fact (0, 1). Сопоставление предикатов fact (N1, N1f) и fact (0, 1) выполняется успешно, так как N1=0, а N1f – свободная переменная. В результате унификации переменная N1f получает значение 1. Так как доказан предикат

Библиотека БГУИР

fact (N1, N1f) (где N1=0, а N1f получила значение 1), доказывается следующий предикат: Nf=N1f*N, где N1f=1 и N=1. Переменная Nf получает значение 1.

Таким образом, доказан предикат fact (N1, N1f), где N1=1, а N1f в результате доказательства получила значение 1. В свою очередь, этот предикат был вызван из тела предиката fact при N=2. Так как предикат fact (N1, N1f) доказан, доказывается следующий предикат: Nf=N1f*N. В результате переменная Nf получает значение 2. Таким образом, доказан предикат fact (N, Nf) с аргументом N=2, вызванный из тела целевого предиката. Переменная Nf получила значение 2. Доказывается следующий предикат (write).

Примечание. Можно сказать, что рекурсивные вызовы предикатов в рекурсивных программах образуют стек.

Необходимо обратить внимание, что для уменьшения переменной на единицу используется запись N1=N-1. Запись типа N=N-1 в Прологе ***недопустима***, так как она рассматривается как предикат *сравнения* N и N-1; очевидно, что он всегда принимает значение "ложь". Присвоить связанной переменной новое значение в Прологе невозможно.

2.2. Списки

Список представляет собой основную структуру данных в Прологе. Список - это последовательность из произвольного числа элементов некоторого типа. Ограничений на длину списка нет. Тип данных "список" объявляется в программе на Прологе следующим образом:

```
domains
списковый_тип = тип*
```

где "тип" - тип элементов списка; это может быть как стандартный тип, так и нестандартный, заданный пользователем и объявленный в разделе domains ранее.

Ниже приведен пример описания предиката, аргументом которого является список целых чисел:

```
domains
int_list = integer*
predicates
nondeterm numbers (int_list)
```

Ввод списков с клавиатуры выполняется стандартным предикатом readterm (<тип>, <имя_переменной>), где <тип> - списковый тип, который должен быть предварительно объявлен в разделе domains. Например, для ввода

Библиотека БГУИР

списка, объявление которого (int_list) показано выше, можно использовать следующий предикат: readterm (int_list, L). При вводе список необходимо набирать точно так же, как он записывается, т.е. в квадратных скобках, через запятую, без пробелов. Если элементы списка - строки, то они должны заключаться в двойные кавычки.

Вывод списка на экран выполняется (в простейшем случае) предикатом write (L), где L - список.

В языке Пролог для удобства обработки списков введены понятия головы и хвоста списка. Голова - это первый элемент списка; хвост - вся остальная часть списка. Таким образом, *хвост списка* сам является *списком*. Для представления списка в виде головы и хвоста используется следующая запись: [H|T], где H - переменная, связываемая с головой (первым элементом) списка, а T - переменная, связываемая с хвостом списка (т.е. также со списком).

Пустой список обозначается так: []. Запись вида [X] обозначает список из одного элемента.

Приведем несколько примеров операций с головой и хвостом списка.

Пусть, например, есть список L=[1,2,3,4]. Тогда предикат L=[H|T] будет доказан успешно (конечно, если H и T - свободные переменные). При этом переменная H связывается с числом 1, а T - со списком [2,3,4]. Предикат L=[X1,X2|T] также доказывается успешно; X1=1, X2=2, T=[3,4].

Пусть имеется список Z=[6,8], а M и N – свободные переменные. Тогда предикат Z=[M|N] доказывается успешно. Переменная M связывается с числом 6, а N – со списком из одного элемента [8]. Важно понимать, что N – именно список из одного числа, но не число. Поэтому, например, операция X=M+1 допустима (в результате переменная X получит значение 7), а Y=N+1 – недопустима: такая запись вызовет ошибку уже при компиляции программы.

Пусть X=7, L=[1,2,3,4]. Результатом операции L1=[X|L] является список [7,1,2,3,4]. Предикат L=[X|L] недопустим (заканчивается неудачей).

Пусть X=7, Y=2, L=[7,5,9], Z – свободная переменная. Тогда предикат L=[X|Z] завершается успешно, так как он заключается в сопоставлении головы списка L с переменной X, и эти величины равны. Предикат L=[Y|Z] заканчивается неудачей, так как голова списка не равна 2.

Пусть L=[5] (список из одного элемента), X и Y – свободные переменные. Тогда предикат L=[X|Y] выполняется успешно. Переменная X связывается со значением 5 (числом), а Y – с пустым списком.

Библиотека БГУИР

Пусть Z=[] (пустой список). Тогда предикат Z=[A|B] заканчивается неудачей (независимо от того, свободны или связаны переменные A и B), так как пустой список нельзя разделить на голову и хвост.

В табл. 2.1 приведены примеры сопоставления предикатов, аргументами которых являются списки. Пусть выполняется сопоставление двух предикатов с именем obrab, аргументы которых - списки целых чисел.

Таблица 2.1

Предикат 1	Предикат 2	Результат
obrab (L) L=[1,2,3]	obrab (X) X – свободная переменная	Успешное сопоставление; X=[1,2,3].
obrab (L) L=[1,2,3]	obrab ([H|T]) H и T – свободные	Успешное сопоставление; H=1, T=[2,3].
obrab (L) L=[2]	obrab ([H|T]) H и T – свободные	Успешное сопоставление; H=2, T=[].
obrab (L) L=[2]	obrab ([X]) X – свободная	Успешное сопоставление; X=2 (здесь X - целочисленная переменная, а не список).
obrab (L) L=[]	obrab ([])	Успешное сопоставление, так как аргументы предикатов равны (оба – пустые списки).
obrab (L) L=[1,2,3]	obrab ([])	Неудача, так как аргументы не равны.

2.3. Пример программы для обработки списков: удаление заданного элемента из списка

Прежде чем рассматривать пример программы для обработки списков, приведем некоторые общие рекомендации по составлению таких программ:

- в операциях со списками практически всегда используется рекурсия;
- обычно предикаты для реализации операций со списками имеют несколько клозов. Первый из них относится к некоторому простейшему случаю (например, к операции с пустым списком, со списком из одного элемента, с головой списка). Последующие клозы относятся к более общим случаям и имеют следующее назначение: если список не соответствует простейшему случаю, рассмотренному в первом клозе, то его следует уменьшить на один элемент (обычно - исключить голову) и применить рекурсию к укороченному списку;
- предикаты, предназначенные для получения одного списка из другого, *всегда* имеют *не менее двух аргументов*: один - исходный список, другой - список-результат.

Библиотека БГУИР

К числу основных операций со списками можно отнести следующие: проверка принадлежности к списку, добавление элемента в список, удаление элемента из списка, присоединение одного списка к другому, вывод элементов списка на экран. На основе этих операций реализуются другие, более сложные операции со списками.

Рассмотрим пример: удаление элемента из списка. Эта операция может быть описана следующим образом. Если удаляемый элемент - голова списка, то результатом операции является его хвост. В противном случае следует выделить хвост списка, удалить из него желаемый элемент и восстановить голову списка.

Программа на Прологе, реализующая эту операцию для списка чисел, имеет следующий вид.

```
domains
int_list=integer*
predicates
nondeterm udal (int_list, integer, int_list)
nondeterm obrab
goal
obrab.
clauses
obrab:- clearwindow,
         write ("Введите список: "), readterm (int_list, List),
         write ("Введите удаляемый элемент: "), readint (X),
        udal (List, X, Lres),
        write (Lres).
udal ([], _, []): -!.
udal (L, X, Lr):- L=[X|T], Lr=T, !.
udal (L, X, Lr):- L=[H|T], udal (T, X, Lr1), Lr=[H|Lr1].
```

Рассмотрим работу предиката udal на примере удаления числа 6 из списка [4,8,6,5]. Вызывается предикат udal (X, List, Lres), где X=6, List=[4,8,6,5], Lres - пока свободная переменная. Попытка сопоставления с первым клозом предиката udal заканчивается неудачей, так как первый аргумент доказываемого предиката - не пустой список. Выполняется сопоставление со вторым клозом udal. При этом переменная L принимает значение [4,8,6,5], X – значение 6; происходит также унификация переменных Lres и Lr. Однако доказательство предиката L=[X|T] завершается неудачей, так как голова списка – число 4, а X=6. Таким образом, попытка доказательства второго клоза

Библиотека БГУИР

предиката udal завершилась неудачей. Происходит возврат и обращение к третьему клозу.

Рассматривается третий клоз предиката udal, где L=[4,8,6,5], X=6. Выделяется голова списка H=4 и хвост T=[8,6,5]. Выполняется рекурсия: вызывается предикат udal (T, X, Lr1), где T=[8,6,5], X=6. Попытки сопоставления с первым и вторым клозами предиката udal неудачны: обращение к первому клозу заканчивается неудачей, так как список T не пустой, а ко второму – так как головой списка T является число 8, а не 6. Снова рассматривается третий клоз. Список L=[8,6,5] разбивается на голову (H=8) и хвост (T=[6,5]). Вызывается предикат udal (T, X, Lr1), где T=[6,5], X=6. Важно понимать, что при этом в памяти *сохраняется* информация о том, что обработка предыдущих вызовов предиката udal еще не закончена. Сопоставление с первым клозом udal заканчивается неудачей, так как список T – не пустой. Выполняется сопоставление со вторым клозом: переменная L получает значение [6,5], а X – значение 6. Предикат L=[X|T] доказывается успешно, так как голова списка L и значение переменной X совпадают (равны 6). Переменная T получает значение [5]. Доказывается предикат Lr=T, в результате Lr=[5]. Таким образом, второй клоз предиката udal доказан. Предикат отсечения (!) нужен только для того, чтобы третий клоз udal не сохранялся в качестве нерассмотренной альтернативы. В данном примере предикат ! не оказывает существенного влияния на работу программы.

Успешно завершенное обращение ко второму клозу предиката udal было выполнено из третьего клоза этого же предиката. Таким образом, доказан предикат udal (T, X, Lr1), где T=[6,5], X=6, а переменная Lr1 получает значение [5] в результате унификации с переменной Lr из заголовка второго клоза.

Доказывается следующий предикат: Lr=[H|Lr1], где H=8, Lr1=[5]. Таким образом, Lr=[8,5]. На этом заканчивается доказательство третьего клоза предиката udal. Он был вызван *из самого себя*. Таким образом, доказан предикат udal (T, X, Lr1), где T=[8,6,5], X=6, а переменная Lr1 получает значение [8,5] в результате унификации с переменной Lr из заголовка третьего клоза.

Так как предикат udal (T, X, Lr1) доказан, доказывается следующий предикат: Lr=[H|Lr1], где H=4, Lr1=[8,5]. Переменная Lr принимает значение [4,8,5]. На этом заканчивается доказательство предиката udal (L, X, Lr), вызванного из целевого предиката. Здесь L=[4,8,6,5], X=6, Lr=[4,8,5].

Таким образом, доказан предикат udal (List, X, Lres) в целевом предикате. Здесь List=[4,8,6,5], X=6, Lres=[4,8,5]. Переменная Lres получила значение в

Библиотека БГУИР

результате унификации с переменной Lr из третьего клоза предиката udal. Так как предикат udal (List, X, Lres) доказан, выполняется следующий предикат (write). На экран выводится список Lres.

Примечание. Если в качестве удаляемого элемента X ввести число, отсутствующее в списке List, то будет получен список Lres, совпадающий с исходным списком List. Такой результат обеспечивается первым клозом предиката udal. Это можно показать, проделав рассуждения, аналогичные приведенным выше.

Примечание. Второй клоз предиката udal можно записать короче: udal ([X|Lr], X, Lr):- !.

2.4. Другие примеры предикатов для обработки списков

Пример. Добавление элемента в конец списка. Операция может быть описана следующим образом: если исходный список пуст, то следует сформировать список из одного элемента, в противном случае - отделить от исходного списка голову, добавить в конец хвоста новый элемент и составить из головы и "увеличенного" хвоста новый список. Добавляемый элемент обозначен как X, исходный список – List, список-результат – Lres.

```
dobav (X, [], [X]):- !.
dobav (X,List, Lres):- List=[H|T], dobav (X, T, T1),Lres=[H|T1].
```

Пример. Определение длины списка. Операция описывается следующим образом: если список пуст, то его длина равна нулю, иначе - длине хвоста плюс единица.

```
dlina ([], 0):-!.
dlina (List, N):- List=[_|T], dlina (T, N1), N=N1+1.
```

Пример. Проверка принадлежности элемента списку. Первый аргумент (переменная X) – элемент, принадлежность которого требуется проверить; второй аргумент – список. Описание операции: проверить, совпадает ли элемент X с головой списка; если не совпадает, то проверить, не принадлежит ли он хвосту списка. Если элемент X не принадлежит списку, то доказательство предиката заканчивается неудачей.

```
prinadl (X, L):- L=[X|_], !.
prinadl (X, L):- L=[_|T], prinadl (X, T).
```

Пример. Поиск максимального элемента списка (переменная Max). Операция описывается следующим образом. Если список пуст, то его максимальный элемент считается равным нулю. Если список состоит из одного элемента, то он и есть максимальный. Если список состоит из нескольких

Библиотека БГУИР

элементов, то следует выбрать максимальный элемент из хвоста списка, а затем сравнить его с головой списка.

```
max_elem ([], Max):- Max=0, !.
max_elem ([X], Max):- Max=X, !.
max_elem ([H|T], Max):- max_elem (T, M), max1 (H, M, Max).

max1 (X1, X2, Max):- X1>=X2, Max=X1, !.
max1 (_, X2, Max):- Max=X2.
```

Здесь предикат max1 просто выбирает максимальное из двух чисел.

Пример. Выделение части (подсписка) из списка. Первый аргумент предиката - список, из которого требуется выделить подсписок; второй и третий - соответственно начало и конец подсписка; четвертый аргумент - результат.

```
podspisok (L, 1, 1, Res):- L=[H|_], Res=[H], !.
podspisok (L, 1, N2, Res):- L=[H|T], N21=N2-1,
                    podspisok (T, 1, N21, Res1), Res=[H|Res1], !.
podspisok (L, N1, N2, Res):- L=[_|T], N11=N1-1, N21=N2-1, podspisok(T,N11,N21,Res).
```

Пример. Перестановка элементов списка в обратном порядке (например, если исходный список – [2,7,5], то результат – [5,7,2]). Первый аргумент предиката – исходный список, второй – результат. Операция может быть описана следующим образом. Если список пуст, то результат – тоже пустой список. В других случаях следует отделить от списка голову, переставить элементы хвоста в обратном порядке и добавить голову в конец полученного списка.

```
perest ([], []):-!.
perest(List, Lres):– List=[H|T], perest (T, Tres), dobav (H, Tres, Lres).
```

Использованный в этом примере предикат dobav показан выше. Данный пример показывает возможность использования предиката для обработки списков (в данном случае – предиката dobav) в другом предикате для обработки списков (perest).

2.5. Примеры программ с использованием списков

Пример. Имеется информация о сотрудниках, работающих над проектами, и характеристики самих проектов (см. лабораторную работу 1). Так как каждый сотрудник может работать над несколькими проектами, для представления такой информации удобно использовать списки. Например,

Библиотека БГУИР

информация о сотруднике Иванове, работающем над проектами П70, П100 и П120, может быть представлена следующим фактом:

rabota (“Иванов”, [“П70”,“П100”,”П120”]).

Требуется составить программу, которая будет запрашивать название предприятия-заказчика и выводить на экран список всех сотрудников, работающих над проектами этого заказчика (с указанием шифров этих проектов).

```
domains
spisok=string*
predicates
nondeterm rabota (string, spisok)
nondeterm proekt (string, string, integer)
nondeterm prinadl (string, spisok)
nondeterm poisk (string)
nondeterm start
goal
start.
clauses
start:- clearwindow, write ("Укажите заказчика: "), readln (Zak), poisk (Zak).
poisk (Zak):- proekt (Shifr, Zak,_), write (Shifr), nl,
              rabota (Fam, Spis_shifr), prinadl (Shifr, Spis_shifr),
              write (Fam), nl, fail.
poisk(_).
prinadl (X, L):- L=[X|_], !.
prinadl (X, L):- L=[_|T], prinadl (X, T).
rabota ("Антонов", ["П20"]).
rabota ("Иванов", ["П70", ”П100”,”П120”]).
rabota ("Петров", ["П100"]).
rabota ("Васильев", ["П70","П120"]).
proekt ("П20", "Рубин", 700).
proekt ("П100", "Рубин", 1400).
proekt ("П70", "Горизонт", 500).
proekt ("П120", "Кристалл", 1100).
```

Рассмотрим работу данной программы для случая, когда введено название заказчика – “Рубин”. После ввода названия заказчика выполняется доказательство предиката poisk. Этот предикат – правило, поэтому требуется доказать все предикаты, входящие в него. Сначала доказывается предикат proekt (Shifr, Zak,_), где Zak=”Рубин”. Для этого он сопоставляется с первым фактом proekt, т.е. с фактом proekt ("П20", "Рубин", 700). Сопоставление выполняется успешно, и переменная Shifr получает значение “П20”. Это

Библиотека БГУИР

значение выводится на экран. Доказывается следующий предикат - rabota (Fam, Spis_shifr). Для этого выполняется его сопоставление с первым фактом rabota, т.е. rabota ("Антонов", ["П20"]). Переменные получают значение Fam=”Антонов”, Spis_shifr=[“П20”]. Доказывается предикат prinadl (Shifr, Spis_shifr), где Shifr=”П20”, Spis_shifr=[“П20”]. Доказательство предиката prinadl выполняется успешно, так как элемент Shifr принадлежит списку Spis_shifr (работа предиката prinadl рассмотрена выше). Доказывается следующий предикат (write), выводящий на экран переменную Fam=”Антонов”.

Стандартный предикат fail вызывает искусственное состояние неудачи. Происходит возврат к ближайшей “развилке”, в данном случае – к предикату rabota (Fam, Spis_shifr). Он сопоставляется со следующим фактом rabota, т.е. с фактом rabota ("Иванов", ["П70",”П100”,”П120”]). Переменные получают значения Fam="Иванов", Spis_shifr=["П70",”П100”,”П120”]. Доказательство предиката prinadl (Shifr, Spis_shifr) завершается неудачей, так как значение переменной Shifr=”П20” отсутствует в списке Spis_shifr. Снова происходит возврат, и рассматривается следующий предикат rabota ("Петров", ["П100"]). Доказательство предиката prinadl снова завершается неудачей, так как элемент Shifr=”П20” отсутствует в списке Spis_shifr=["П100"]. Снова выполняется возврат, рассматривается предикат rabota ("Васильев", ["П70","П120"]), однако доказательство предиката prinadl снова завершается неудачей, как и в предыдущем случае.

Так как все факты rabota рассмотрены, происходит возврат к следующей “развилке”: к предикату proekt (Shifr, Zak,_), где Zak=”Рубин”. Он сопоставляется с фактом proekt ("П100", "Рубин", 1400). Сопоставление выполняется успешно, переменная Shifr получает значение “П100”, и оно выводится на экран. Доказывается следующий предикат - rabota (Fam, Spis_shifr). Он сопоставляется с первым фактом rabota, т.е. rabota ("Антонов", ["П20"]). Доказывается предикат prinadl (Shifr, Spis_shifr), где Shifr=”П100”, Spis_shifr=[“П20”]. Доказательство предиката prinadl завершается неудачей, так как элемент Shifr отсутствует в списке Spis_shifr. Снова происходит возврат, предикат rabota (Fam, Spis_shifr) сопоставляется с фактом rabota ("Иванов", ["П70",”П100”,”П120”]). Предикат prinadl rabota (Fam, Spis_shifr) доказывается успешно, так как значение Shifr=”П100” принадлежит списку Spis_shifr=["П70",”П100”,”П120”]. Следующий предикат (write) выводит на экран переменную Fam=”Иванов”.

После этого снова возникает искусственная неудача (fail). Происходит возврат к предикату rabota (Fam, Spis_shifr); он сопоставляется со следующим

Библиотека БГУИР

фактом rabota ("Петров", ["П100"]). Предикат prinadl (Shifr, Spis_shifr) доказывается успешно, как и в предыдущем случае. На экран выводится переменная Fam=”Петров”.

Предикат fail вызывает искусственный возврат. Происходит сопоставление предиката rabota (Fam, Spis_shifr) с фактом rabota ("Васильев", ["П70", "П120"]). Доказательство предиката prinadl завершается неудачей, так как элемент “П100” отсутствует в списке ["П70", "П120"]. Происходит возврат к предикату proekt (Shifr, Zak,_), так как все клозы предиката rabota уже рассмотрены. Попытки сопоставления предиката proekt (Shifr, Zak,_) с фактами proekt ("П70", "Горизонт", 500) и proekt ("П120", "Кристалл", 1100) неудачны из-за несовпадения второго аргумента (так как Zak=”Рубин”). Доказывается второй клоз предиката poisk. На этом завершается доказательство целевого предиката start и выполнение программы в целом.

Пример. Требуется составить программу, которая будет запрашивать список названий предприятий-заказчиков и выводить для каждого из них список всех сотрудников, работающих над проектами этого заказчика, и шифры этих проектов.

```
domains
spisok=string*
predicates
nondeterm rabota (string, spisok)
nondeterm proekt (string, string, integer)
nondeterm prinadl (string, spisok)
nondeterm poisk (string)
nondeterm obrab_spisok (spisok)
nondeterm start
goal
start.
clauses
start:- clearwindow, write ("Введите список заказчиков: "),
      readterm (spisok, Spis_zak), obrab_spisok (Spis_zak).
obrab_spisok (L):- L=[], !.
obrab_spisok (L):- L=[H|T], poisk (H), obrab_spisok (T).
/* Остальная часть программы такая же, как в предыдущем примере */
```

Предикат readterm выполняет ввод переменной Spis_zak; тип этой переменной – spisok. Например, если требуется получить список сотрудников, работающих над проектами предприятий “Рубин” и “Кристалл”, то по запросу

Библиотека БГУИР

“Введите список заказчиков: “ необходимо ввести [“Рубин”,”Кристалл”] (без пробелов). Тип spisok объявлен в разделе domains и представляет собой список строк.

При доказательстве предиката obrab_spisok сначала происходит сопоставление с его первым клозом, однако оно заканчивается неудачей (так как введенный список заказчиков – не пустой). Поэтому рассматривается второй клоз. Список заказчиков L разделяется на голову (H=”Рубин”) и хвост (T=[“Кристалл”]). Для переменной H=”Рубин” выполняется предикат poisk, подробно рассмотренный выше. При рекурсивном вызове предиката obrab_spisok (T), где T=[“Кристалл”], сначала рассматривается первый клоз предиката obrab_spisok. Однако доказательство первого клоза снова заканчивается неудачей, так как список T не пустой. Поэтому снова рассматривается следующий (второй) клоз. При сопоставлении предикатов obrab_spisok (L) и obrab_spisok (T) переменная L получает значение [“Кристалл”]. Список L разделяется на голову (H=”Кристалл”) и хвост (T=[], т.е. пустой список). Для переменной H=”Кристалл” выполняется предикат poisk. Снова выполняется рекурсивный вызов предиката obrab_spisok (T), где T=[]. При сопоставлении предиката obrab_spisok (T) с первым клозом obrab_spisok (L) переменная L унифицируется с переменной T, т.е. получает значение пустого списка. Поэтому условие L=[] выполняется, и предикат obrab_spisok успешно доказывается.

Порядок выполнения работы

Отладить в среде Visual Prolog программу, выполняющую операции по обработке списков (по заданию, выданному преподавателем).

Контрольные вопросы

1. Понятие списка. Объявление списков.
2. Разделение списка на голову и хвост.
3. Ввод и вывод списков.
4. Примеры операций по обработке списков.
5. Принцип работы программ на Прологе, выполняющих обработку списков (на примере программы, отлаженной в ходе выполнения работы).

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №3

РАЗВИТЫЕ ТИПЫ ДАННЫХ В ПРОГРАММАХ НА ПРОЛОГЕ. БАЗЫ ДАННЫХ В ПРОГРАММАХ НА ПРОЛОГЕ

Цель работы. Изучение развитых типов данных в программах на Прологе, необходимых для программной реализации экспертных систем. Изучение возможностей построения и обработки баз данных на Прологе.

3.1. Функторы и альтернативные домены

Функторы представляют собой составные объекты программы на Прологе. Они позволяют структурировать данные для удобства их представления и обработки. Функторы объявляются в разделе domains следующим образом:

```
domains
тип = имя_функтора (типы аргументов)
```

Одна из наиболее важных задач, для которых применяются функции в программах на Прологе – объявление аргументов предикатов, которые в разных случаях могут принимать значения разных типов. В этом случае необходимо объявить набор функторов, называемых *альтернативным доменом*:

```
domains
альтернативный_домен = функтор_1 (типы аргументов);
                       функтор_2 (типы аргументов);
                       .....
                       функтор_n (типы аргументов)
```

Если затем указать "альтернативный домен" в качестве типа аргумента при объявлении предиката (в разделе predicates), то в качестве аргумента предиката можно будет указывать любой из функторов "функтор_1", ..., "функтор_n".

Пример. Имеется информация о проектах и о сотрудниках, работающих над ними (см. лабораторную работу 2). Предположим, что стоимость контрактов на разработку всех проектов указывается в тысячах долларов, но по некоторым проектам оплата производится в рублях по курсу на определенную дату. Для проектов, оплачиваемых в рублях, требуется указывать не только стоимость контракта (в долларах), но и дату для определения курса. Для проектов, оплачиваемых в долларах, указывать дату не требуется. Предположим, что оплата по проектам П70 и П100 производится в долларах, а

Библиотека БГУИР

по проектам П20 и П120 – в рублях. Тогда факты базы данных, описывающие проекты, могут иметь следующий вид:

```
proekt ("П20", "Рубин", rub(700,”10.09.2002”)).
proekt ("П100", "Рубин", doll(1400)).
proekt ("П70", "Горизонт", doll(500)).
proekt ("П120", "Кристалл", rub(1100,”12.10.2002”)).
```

Здесь, например, указано, что стоимость контракта по проекту П120 составляет 1 млн 100 тыс. долларов, но оплата будет производиться в рублях по курсу на 12 октября 2002 г.

Здесь rub и doll – функторы. Конечно, можно использовать и любые другие имена. Альтернативный домен, включающий функторы rub и doll, должен быть объявлен следующим образом:

```
domains
oplata=rub (integer, string);
        doll (integer)
```

Здесь oplata – имя альтернативного домена (можно использовать и любое другое имя).

Предикат proekt, в котором используется альтернативный домен oplata, должен быть объявлен в разделе predicates следующим образом:

```
predicates
proekt (string, string, oplata)
```

Такое объявление означает, что третьим аргументом предиката proekt может быть любой функтор, входящий в альтернативный домен oplata, т.е. rub или doll.

Примечание. При работе с функторами важно не путать их с предикатами. Функтор - не предикат, а элемент данных; он является аргументом предиката. Объединение нескольких элементов данных с помощью функтора *никогда не означает* какой-либо операции над ними. Стандартных функторов, а также зарезервированных слов, связанных с функторами, в Прологе *нет*.

Ввод функтора может выполняться стандартным предикатом readterm (тип, имя_переменной). Здесь "тип" – альтернативный домен, указанный в разделе domains (для рассмотренного примера - oplata). "Имя_переменной" - переменная, с которой связывается функтор. Функтор вводится в точном соответствии с его объявлением: указывается имя функтора (в рассмотренном примере – rub или doll) и его аргументы. Аргументы функтора указываются в скобках, без пробелов, разделяются запятыми. Строковые аргументы вводятся в кавычках.

Пример. Программа для вывода на экран перечня всех проектов, оплачиваемых в рублях (с указанием заказчика и стоимости контракта).

Библиотека БГУИР

```
domains
oplata=rub (integer, string);
        doll (integer)
predicates
nondeterm proekt (string, string, oplata)
nondeterm poisk
goal
poisk.
clauses
poisk:- proekt (Shifr, Zak, rub (Stoim,_)),
         write (Shifr, "  ", Zak, "  ", Stoim), nl, fail.
poisk.
proekt ("П20", "Рубин", rub(700,”10.09.2002”)).
proekt ("П100", "Рубин", doll(1400)).
proekt ("П70", "Горизонт", doll(500)).
proekt ("П120", "Кристалл", rub(1100,”12.10.2002”)).
```

При доказательстве целевого предиката poisk сопоставление предиката proekt (Shifr, Zak, rub (Stoim,_)) с фактами proekt выполняется успешно только в том случае, если третьим аргументом факта proekt является функтор rub. Для вывода всех желаемых проектов используется механизм искусственного возврата (fail), рассмотренный в лабораторной работе 1.

3.2. Базы данных в программах на Прологе

В программах на Прологе имеется возможность хранить предикаты-факты, обрабатываемые программой, отдельно от самой программы. Файл, содержащий предикаты-факты, называется базой данных.

Факты, хранящиеся в файле базы данных, объявляются в разделе global facts (или global database). Нестандартные типы данных, используемые в базе данных, должны быть объявлены в разделе global domains.

В одной базе данных могут храниться факты, имеющие разные имена, разное количество аргументов и т.д. Все они должны быть объявлены в разделе global facts.

Пример. Чтобы хранить в файле базы данных информацию о проектах и о сотрудниках, работающих над ними (см. предыдущий пример, а также пример из лабораторной работы 2), потребуется следующее объявление:

```
global domains
oplata=rub (integer, string);
        doll (integer)
spisok=string*
global facts
```

Библиотека БГУИР

rabota (string, spisok)
proekt (string, string, oplata)

Информация в базу данных может вводиться обычным текстовым редактором (например, редактором системы Visual Prolog). В каждой строке базы данных может содержаться *только один факт*. В базе данных не должно быть пустых строк, *в том числе в конце базы данных (после последнего факта)*. Факты в базе данных должны быть набраны *без пробелов*. Точка в конце фактов *не ставится*.

3.3. Загрузка базы данных

Загрузка базы данных во время работы программы выполняется стандартным предикатом consult (имя_файла). Имя файла базы данных может задаваться константой (в этом случае оно должно быть заключено в кавычки) или переменной. Если используется переменная, то ей должно быть присвоено имя файла базы данных. Для указания имени файла в этом случае может использоваться предикат readln или операция присваивания.

Если имя файла базы данных *указывается в тексте программы* (например, задается в предикате consult или присваивается переменной), то в нем необходимо использовать двойные наклонные черты (\\). Если имя файла *вводится с клавиатуры* с помощью оператора readln, то используются одиночные наклонные черты (\).

Пусть, например, база данных, содержащая информацию о проектах и о сотрудниках, работающих над этими проектами, хранится в файле PROEKT.DBA на диске E: в папке ES. Тогда для загрузки базы данных можно использовать следующий предикат: consult(“e:\\es\\proekt.dba”).

Перед предикатом consult рекомендуется указывать стандартный предикат retractall(_), удаляющий из памяти все факты базы данных (например, если они сохранились от предыдущего запуска программы). Подробнее этот предикат будет рассмотрен в подразделе 3.5.

Пример. Предикат, запрашивающий имя базы данных и выполняющий ее загрузку.

```
zagruzka:- write(“Введите имя базы данных: ”), readln(F), retractall(_), consult(F), !.
zagruzka.
```

По запросу “Введите имя базы данных” может быть введено, например, следующее имя: e:\es\proekt.dba.

Второй клоз предиката zagruzka, не выполняющий никаких действий, предусмотрен на случай, если пользователь откажется от загрузки, нажав клавишу ESC. В этом случае стандартный предикат readln завершается

Библиотека БГУИР

неудачей. Происходит возврат ко второму клозу предиката zagruzka, и он доказывается успешно.

3.4. Поиск информации в базе данных

Прежде чем выполнять какие-либо действия по поиску информации в базе данных, необходимо загрузить ее в память компьютера, используя стандартный предикат consult. После этого предикаты-факты, загруженные из файла базы данных, могут использоваться в программе точно так же, как и факты, указанные в самой программе.

Пример. Предикаты для просмотра перечня всех сотрудников, работающих над проектами:

```
prosmotr_sotr:- rabota (F, S),
                write (“Фамилия: ”, F,"  Проекты: ",S), nl, readchar(_), fail.
prosmotr_sotr.
```

Работа этого предиката ничем не отличается от работы аналогичных предикатов, рассмотренных в лабораторной работе 1 (подразделы 1.5, 1.6.1).

3.5. Изменение базы данных

В ходе работы программы можно создавать новые факты базы данных или, наоборот, удалять имеющиеся. Поэтому базы данных в программах на Прологе называются *динамическими*.

Для создания новых фактов используются предикаты asserta (добавление факта в начало базы данных), assertz (добавление факта в конец базы данных), assert (действует так же, как assertz).

Пример. Предикат для добавления информации о новом сотруднике. Вводится фамилия сотрудника и список проектов, над которыми он работает.

```
dobav_sotr:- write ("Фамилия: "), readln (Fam),
             write ("Проекты: "), readterm (spisok, Spis_pr),
             assert (rabota (Fam, Spis_pr)).
```

Предикат assert создает в памяти новый факт rabota. Например, если введена фамилия Сергеев и список проектов [“П100”,”П120”], то будет создан следующий факт: rabota(“Сергеев”,[“П100”,”П120”]).

Аналогично можно реализовать предикат для добавления информации о проекте:

```
dobav_pr:- write ("Шифр: "), readln (Shifr),
           write ("Заказчик: "), readln (Zak),
           write ("Оплата: "), readterm(oplata, Opl),
           assert (proekt(Shifr, Zak, Opl)).
```

Библиотека БГУИР

Следует напомнить, что в рассматриваемом примере третий аргумент предиката proekt (оплата) задается с помощью *функтора*. Например, если для некоторого проекта требуется указать, что его стоимость составляет 800 тыс. долларов и оплачивается в долларах, то по запросу “Оплата:” необходимо ввести: doll(800). Если проект стоимостью 800 тыс. долларов оплачивается в рублях по курсу на 5 сентября 2002 года, то следует ввести: rub(800,”5.09.2002”).

Для удаления фактов базы данных используются стандартные предикаты retract и retractall.

Пример. Предикат для удаления информации о сотруднике.

```
udal_sotr:- write ("Фамилия: "), readln (Fam), retractall (rabota (Fam,_)).
```

Предикат retractall удаляет *все* факты, соответствующие указанным данным. В данном примере удаляются все факты rabota, где первым аргументом является указанная фамилия. Если таких фактов нет (например, фамилия введена неверно), то никаких изменений в базе данных не происходит, однако предикат rertractall считается доказанным успешно.

Предикат retractall(_) удаляет из памяти *все факты базы данных*. Его рекомендуется указывать перед загрузкой базы данных, чтобы удалить из памяти факты, которые могли сохраниться от предыдущего запуска программы.

Предикат retract удаляет из памяти *только первый* факт, соответствующий указанным данным. Если таких фактов нет, то никаких изменений в базе данных не происходит; однако доказательство предиката rertractall считается *неудачным*, и происходит *возврат*. Поэтому чаще применяется предикат retractall.

Примечание. Важно обратить внимание, что все рассмотренные возможности изменения базы данных относятся только к фактам, загруженным *в память* компьютера, но не к содержимому *файла* базы данных. Например, если удалить какие-либо факты с помощью предикатов retract или retractall, то они удаляются только из памяти, но не из файла, хранящегося на диске. Чтобы изменить содержимое файла базы данных, необходимо сначала внести необходимые изменения в памяти, а затем сохранить базу данных, как показано ниже.

3.6. Сохранение базы данных

Для сохранения содержимого базы данных в файле используется стандартный предикат save (имя_файла). Например, чтобы сохранить базу данных в файле PROEKT.DBA на диске E: в папке ES, можно использовать предикат: save(“e:\\es\\proekt.dba”). Можно также запрашивать имя базы данных, как показано в следующем примере.

Библиотека БГУИР

Пример. Предикат, запрашивающий имя файла и выполняющий сохранение базы данных в указанном файле.

```
sohran:- write("Введите имя базы данных для сохранения: "), readln(F), save(F), !.
sohran.
```

При вводе путевого имени файла необходимо использовать одиночные наклонные черты (\).

Второй клоз предиката sohran обеспечивает его успешное доказательство в случае, если пользователь откажется от сохранения, нажав клавишу ESC.

3.7. Дополнительные возможности применения баз данных

Динамические базы данных широко применяются в Прологе для самых разнообразных операций по обработке данных. Предикаты assert и retract (или retractall) позволяют организовать временное хранение и изменение данных.

Пример. Предикат, вычисляющий суммарную стоимость проектов, оплачиваемых в рублях и в долларах.

```
summa:- retractall (sumdoll(_)), assert (sumdoll (0)),
        proekt (_, _, doll(S)), sumdoll (Sd), Sd1=Sd+S,
        retractall(sumdoll(_)), assert(sumdoll(Sd1)), fail.
summa:- retractall (sumrub(_)), assert (sumrub (0)),
        proekt (_, _, rub(S,_)), sumrub (Sr), Sr1=Sr+S,
        retractall (sumrub(_)), assert (sumrub(Sr1)), fail.
summa:- sumdoll (Sd), write ("Оплата в долларах: ", Sd), nl,
        sumrub (Sr), write ("Оплата в рублях: ", Sr), readchar(_).
```

При вызове этого предиката сначала происходит обращение к первому клозу. Стандартный предикат retractall (sumdoll(_)) удаляет из памяти факт базы данных sumdoll, если он там есть. Стандартный предикат assert (sumdoll(0)) создает факт базы данных sumdoll(0). Затем в базе данных отыскивается первый предикат proekt, где третьим аргументом является функтор doll; это предикат proekt ("П100", "Рубин", doll(1400)). Переменная S принимает значение 1400. В базе данных отыскивается предикат sumdoll(0), и переменная Sd принимает значение 0. Переменная Sd1 принимает значение Sd+S=0+1400=1400. Предикат retractall(sumdoll(_)) удаляет из памяти факт sumdoll(0), а предикат assert – создает в памяти предикат sumdoll(1400). Предикат fail создает состояние искусственной неудачи; это требуется, чтобы найти сумму стоимостей по *всем* проектам.

Происходит возврат к ближайшей "развилке", в данном случае – к предикату proekt (_, _, doll(S)). В базе данных отыскивается следующий предикат proekt, где третьим аргументом является функтор doll; это предикат proekt ("П70", "Горизонт", doll(500)). Переменная S принимает значение 500.

Библиотека БГУИР

В базе данных отыскивается предикат sumdoll. Так как имеется предикат sumdoll(1400), переменная Sd принимает значение 1400. Вычисляется переменная Sd1=Sd+S=1400+500=1900. Предикат retractall(sumdoll(_)) удаляет из памяти факт sumdoll(1400), а предикат assert – создает в памяти предикат sumdoll(1900). Предикат fail снова создает состояние искусственной неудачи. Так как в базе данных больше нет фактов proekt с функторами doll, происходит переход ко второму клозу предиката summa. Итак, по окончании обработки первого клоза предиката summa в базе данных создан факт sumdoll(1900). Таким образом, подсчитана сумма стоимостей проектов, оплачиваемых в долларах.

Второй клоз предиката summa обрабатывается аналогично первому. В результате создается факт sumrub(1800), содержащий сумму стоимостей проектов, оплачиваемых в рублях.

В третьем клозе предиката summa выполняется вывод результатов на экран.

Факты базы данных sumdoll и sumrub должны быть объявлены в разделе global facts:

```
global facts

sumdoll (real)
sumrub (real)
```

Примечание. Напомним, что в Прологе невозможны операции вида S=S+X. Такая операция интерпретируется как сравнение значений S и S+X и поэтому всегда завершается неудачей. Из-за этого для реализации суммирования потребовалось использовать факты динамической базы данных.

3.8. Программа для работы с базой данных

Рассмотрим разработку программы, позволяющей выполнять различные операции с базой данных о проектах и сотрудниках.

Объявление типов данных (раздел global domains) и фактов базы данных (раздел global facts) рассмотрено в подразделе 3.2.

Приведем целевой предикат, создающий простейшее меню операций с базой данных и обеспечивающий выполнение выбранных операций.

```
goal

start

clauses

start:-clearwindow,
      makewindow (1,7,7,"База данных",0,0,14,40),
      repeat, clearwindow,
      write ("1 - загрузка"), nl,
```

Библиотека БГУИР

```
write ("2 - просмотр сотрудников"), nl,
write ("3 - просмотр проектов"), nl,
write ("4 - добавление сотрудников"), nl,
write ("5 - добавление проектов"), nl,
write ("6 - удаление сотрудников"), nl,
write ("7 - удаление проектов"), nl,
write ("8 - подсчет сумм"), nl,
write ("9 - сохранение"), nl,
write ("99 - выход"), nl,
write ("Введите номер: "), readint (N),
obrab (N), removewindow.
```

Стандартный предикат makewindow создает на экране окно. Смысл аргументов предиката следующий: 1 – номер окна; 7, 7 – цвет окна и рамки (черное окно, белая рамка), "База данных" – заголовок окна (выводится сверху); 0, 0 – координаты левого верхнего угла окна; 14, 40 – высота и ширина окна.

Предикат repeat необходим для того, чтобы операции с базой данных могли выполняться многократно. Этот предикат рассмотрен в п.1.6.2.

Предикат clearwindow очищает окно. Предикаты write выводят на экран элементы меню (возможные операции с базой данных), а предикат readint запрашивает номер желаемой операции.

Предикат obrab реализует операции, выбираемые из меню. Он должен иметь по меньшей мере десять клозов (по числу операций в меню). При этом все клозы предиката obrab (кроме клоза для операции выхода, соответствующего номеру 99) должны заканчиваться *неудачей*. В этом случае выполняется возврат к предикату repeat. Он успешно доказывается (см. п. 1.6.2), на экран снова выводится меню, и повторяется запрос желаемой операции. Реализация предиката obrab показана ниже.

```
obrab(1):- makewindow(2,7,7,"Загрузка",12,0,3,60), zagruzka,
           removewindow, shiftwindow(1), !,fail.
obrab(2):- makewindow(3,7,7,"Сотрудники",12,0,10,80), prosmotr_sotr,
           removewindow, shiftwindow(1), !,fail.
obrab(3):- makewindow(4,7,7,"Проекты",12,0,10,80), prosmotr_pr,
           removewindow, shiftwindow(1), !,fail.
obrab(4):- makewindow(5,7,7,"Новый сотрудник",12,0,10,80), dobav_sotr,
           removewindow, shiftwindow(1), !,fail.
obrab(5):- makewindow(6,7,7,"Новый проект",12,0,10,80), dobav_pr,
           removewindow, shiftwindow(1), !,fail.
obrab(6):- makewindow(7,7,7,"Удаление сотрудника",12,0,10,80), udal_sotr,
           removewindow,shiftwindow(1), !,fail.
obrab(7):- makewindow(8,7,7,"Удаление проекта",12,0,10,80), udal_pr,
           removewindow, shiftwindow(1), !,fail.
obrab(8):- makewindow(9,7,7,"Подсчет сумм",12,0,10,80),  summa,
           removewindow,shiftwindow(1), !,fail.
obrab(9):- makewindow(10,7,7,"Сохранение",12,0,3,60), sohran,
           removewindow, shiftwindow(1), !,fail.
obrab(99).
```

Библиотека БГУИР

Рассмотрим клоз obrab(2). Он будет доказываться, если на запрос "Введите номер: " будет введен номер 2, т.е. выбрана операция просмотра перечня сотрудников. Предикат makewindow создает окно для просмотра. Затем вызывается предикат prosmotr_sotr, непосредственно выполняющий просмотр перечня сотрудников. Этот предикат рассмотрен в подразделе 3.4. Для нормальной работы программы он должен доказываться успешно. Предикат removewindow удаляет с экрана текущее окно (окно просмотра). Предикат shiftwindow(1) выполняет возврат курсора в окно с номером 1 (окно меню). Предикат fail вызывает состояние искусственной неудачи, в результате чего происходит возврат к предикату repeat и повторение работы с меню. Предикат отсечения (!) указан только для того, чтобы при возврате не рассматривались остальные клозы предиката obrab.

Предикаты zagruzka, prosmotr_sotr, dobav_sotr, dobav_pr, udal_sotr, summa и sohran рассмотрены выше. Предикат prosmotr_pr (просмотр проектов) полностью аналогичен предикату prosmotr_sotr, а udal_pr (удаление проекта) – предикату udal_sotr. Все используемые предикаты должны быть объявлены в разделе predicates.

Порядок выполнения работы

По заданию, выданному преподавателем, подготовить файл базы данных, содержащий несколько фактов. В структуре фактов базы данных должны быть использованы данные типа “список”, ”функтор”, ”альтернативный домен”. Разработать и отладить программу, реализующую следующие операции с базой данных: загрузку, просмотр базы данных, добавление фактов в базу данных, удаление фактов из базы данных (по заданному условию), сохранение базы данных. Программа должна иметь меню и систему окон.

Контрольные вопросы

1. Понятие функтора.
2. Понятие альтернативного домена.
3. Базы данных в программах на Прологе. Объявление баз данных.
4. Основные операции с базами данных.
5. Загрузка и сохранение базы данных.
6. Просмотр и поиск информации в базе данных.
7. Простейшее меню и система окон в программах на Прологе.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №4

ПРЕДСТАВЛЕНИЕ И ОБРАБОТКА ЗНАНИЙ С ИСПОЛЬЗОВАНИЕМ ЛОГИЧЕСКИХ ФУНКЦИЙ

Цель работы. Изучение возможностей представления и обработки знаний в экспертных системах с помощью логических функций

4.1. Представление знаний с помощью логических функций

Во многих случаях знания человека, выраженные в форме высказываний, могут быть представлены с помощью функций логических переменных (булевых функций). Напомним, что логическая (булева) переменная – это переменная, принимающая только два значения: “истина” (1) или “ложь” (0). Такие переменные соответствуют высказываниям, описывающим знания человека.

Из всех известных функций логических переменных, для представления знаний в форме высказываний обычно достаточно использовать функции отрицания, дизъюнкции (логическое “или”), конъюнкции (логическое “и”), импликации.

Отрицание логической переменной А ($\overline{A}$, “не А”) принимает значение “истина”, если переменная А имеет значение “ложь” (высказывание А ложно). Таким образом, $\overline{A}=1$, если А=0, и наоборот.

Дизъюнкция $A \vee B$ (“А или В”) истинна, если *хотя бы одна* из переменных имеет значение “истина” (хотя бы одно из высказываний А или В истинно).

Конъюнкция А&В (“А и В”) истинна, если *обе* переменные имеют значения “истина” (оба высказывания А и В истинны).

Импликация $A \rightarrow B$ (“А влечет В”, “из А следует В”) *ложна*, если переменная А имеет значение “истина”, а В – “ложь” (высказывание А истинно, а В – ложно).

В таблице 4.1 приведены значения функций дизъюнкции, конъюнкции и импликации при различных значениях аргументов. Такая таблица называется таблицей истинности.

Таблица 4.1

A	B	$A \vee B$	A&B	$A \rightarrow B$
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	1

Библиотека БГУИР

В некоторых случаях для упрощения выражений с логическими функциями удобно использовать следующие равенства:

$\overline{A \vee B} = \overline{A} \,\&\, \overline{B}$

$\overline{A \,\&\, B} = \overline{A} \vee \overline{B}$

$A \rightarrow B = \overline{A} \vee B$

Эти равенства легко проверить, составив таблицу истинности для левой и правой части каждого из них.

Пример. Пусть при некотором заболевании могут использоваться лекарства A, B и C. Имеются следующие сведения:

•если используется лекарство A и не используется B, то необходимо использовать лекарство C;

•если используется лекарство C, то необходимо использовать хотя бы одно из лекарств A или B;

•лекарства A и C несовместимы.

Введем логические переменные A, B и C. Пусть эти переменные принимают значение “истина”, если соответствующее лекарство используется, и “ложь” – если не используется. Приведенные выше указания об использовании лекарств можно записать с помощью логических функций следующим образом:

$(A \& \overline{B}) \rightarrow C$

$C \rightarrow (A \vee B)$

$A \rightarrow \overline{C}$

4.2. Представление логических функций в алгебраической форме

Для удобства компьютерной обработки во многих случаях желательно перейти от представления знаний с помощью логических функций к их записи в алгебраической форме, т.е. с помощью обычных переменных, над которыми можно выполнять операции сложения, умножения, сравнения и т.д. Основные правила замены логических функций на алгебраические выражения приведены в табл.4.2.

Таблица 4.2

Логическая функция	Алгебраическое выражение
Отрицание ($\overline{A}$)	1-A
Дизъюнкция ($A \vee B$)	A+B-A·B
Конъюнкция (A&B)	A·B
Импликация ($A \rightarrow B$)	B-A $\geq$ 0

Библиотека БГУИР

Покажем возможность замены логической операции отрицания ($\overline{A}$) на алгебраическое выражение 1-A. Пусть A – логическая переменная. Тогда, по определению операции отрицания, $\overline{A}$=1, если A=0, и $\overline{A}$=0, если A=1. Пусть A – обычная алгебраическая переменная. Тогда 1-A=1, если A=0, и 1-A=0, если A=1. Таким образом, при подстановке одинаковых значений переменной A, выражения $\overline{A}$ и 1-A принимают одинаковые значения.

Покажем возможность замены дизъюнкции ($A \vee B$) на алгебраическое выражение $A+B-A \cdot B$. По определению операции дизъюнкции, $A \vee B$=1, если хотя бы одна из переменных (A или B) принимает значение 1. Если A=0 и B=0, то $A \vee B$=0. Выражение $A+B-A \cdot B$ также принимает значение 1, если хотя бы одна из переменных A или B принимает значение 1. Если A=0 и B=0, то $A+B-A \cdot B=0$.

Аналогично доказывается возможность замены конъюнкции (A&B) на выражение $A \cdot B$.

Рассмотрим замену импликации ($A \rightarrow B$) на условие $B-A \geq 0$. По определению операции импликации, $A \rightarrow B$ – ложно, если A=1, а B=0; в остальных случаях $A \rightarrow B$ – истинно. Рассмотрим условие $B-A \geq 0$ при различных значениях A и B. Пусть A=1, B=0; тогда B-A=-1, и условие $B-A \geq 0$ ложно. Пусть A=0, B=0; тогда условие $B-A \geq 0$ истинно (так как B-A=0). Аналогично можно показать, что условие $B-A \geq 0$ истинно, если A=0, B=1, или A=1, B=1. Таким образом, логическая функция $A \rightarrow B$ и условие $B-A \geq 0$ истинны (или, наоборот, ложны) при одних и тех же значениях переменных A и B.

В таблице 4.3 приведены еще некоторые правила представления условий, налагаемых на логические переменные.

Таблица 4.3

Условие	Алгебраическое представление
Хотя бы одна из логических переменных $X_1, X_2, ..., X_n$ принимает значение “истина”	$X_1+X_2+...+X_n \geq 1$
Только одна из логических переменных $X_1, X_2, ..., X_n$ принимает значение “истина”	$X_1+X_2+...+X_n=1$
Все логические переменные $X_1, X_2, ..., X_n$ одновременно принимают значение “истина”	$X_1+X_2+...+X_n=n$

Пример. Представим в алгебраической форме условия выбора лекарств (см. подраздел 4.1).

Рассмотрим условие $(A \& \overline{B}) \rightarrow C$. Воспользуемся правилами замены, приведенными в табл.4.2. Используя правило замены отрицания, получим:

Библиотека БГУИР

(A&(1-B))→C. Выполнив замену конъюнкции, получим: (A·(1-B))→C. По правилу замены импликации получим: C-(A·(1-B)) ≥ 0.

Примечание. Промежуточные выражения (A&(1-B))→C и (A·(1-B))→C не являются вполне правильными с математической точки зрения, так как в них “смешаны” логические и алгебраические операции. Эти выражения приведены только для иллюстрации процесса перехода от логического к алгебраическому выражению.

Запишем в алгебраической форме условие $C \to (A \vee B)$. Выполнив замену дизъюнкции, получим: C→(A+B-A·B). По правилу замены импликации получим: A+B-A·B-C ≥ 0.

Для условия $A \to \overline{C}$, используя правила замены отрицания и импликации, получим: 1-C-A ≥ 0.

Требуется также указать, что необходимо выбрать хотя бы одно лекарство. Для этого запишем условие: A+B+C ≥ 0.

Таким образом, условия выбора лекарств могут быть записаны в алгебраической форме следующим образом:

C-(A·(1-B)) ≥ 0
A+B-A·B-C ≥ 0
1-C-A ≥ 0
A+B+C ≥ 0

Чтобы выбрать подходящие лекарства (одно или несколько), требуется найти значения переменных A, B, C, удовлетворяющие этим условиям и принимающие значения 0 или 1.

4.3. Программа на языке Пролог для решения логической задачи

Приведем программу на Прологе, реализующую выбор лекарств, в соответствии с указанными выше условиями.

```
predicates

nondeterm poisk
nondeterm znach (integer)

goal

poisk

clauses

poisk:- znach(A), znach (B), znach (C),
        C-(A*(1-B)) >= 0,
        A+B-A*B-C >= 0,
        1-C-A >= 0,
        A+B+C >= 1,
        write ("A=", A, "  B=", B, "  C=", C), !.
poisk:- write ("Заданы противоречивые условия").

znach(X):- X=0.
znach(X):- X=1.
```

Библиотека БГУИР

При доказательстве целевого предиката poisk сначала происходит сопоставление предикатов znach (A), znach (B) и znach (C) с первым клозом предиката znach, т.е. znach (X):- X=0. В результате переменные A, B, C принимают значение 0. Затем происходит проверка условий. Для значений A=0, B=0, C=0 не выполняется условие $A+B+C \geq 1$. Поэтому происходит возврат к ближайшей “развилке”; в данном случае это предикат znach (C). Происходит сопоставление предиката znach (C) со вторым клозом предиката znach. В результате переменная C принимают значение 1. Снова выполняется проверка условий. Процесс повторяется, пока не будут найдены значения переменных A, B, C, удовлетворяющие всем четырем условиям. Эти значения выводятся на экран. Предикат отсечения (!) указан только для того, чтобы второй клоз предиката poisk не сохранялся в памяти как нерассмотренная альтернатива. На выполнение данной программы предикат отсечения не влияет.

Выполнив программу, получим: A=0, B=1, C=0. Таким образом, следует использовать лекарство B.

Примечание. Если бы условия задачи оказались противоречивыми, и никакие значения A, B, C не удовлетворяли бы всем условиям, то доказательство первого клоза предиката poisk завершилось бы неудачей. В этом случае был бы доказан второй клоз предиката poisk, выводящий на экран соответствующее сообщение.

4.4. Решение логических задач с использованием языка Пролог: заключительный пример

Пример. Требуется выполнить три вида работ (Р1, Р2, Р3). Имеется пять исполнителей (И1, И2, И3, И4, И5). При распределении исполнителей по работам необходимо учесть следующее:

- для работы Р1 требуются два исполнителя, для Р2 и Р3 – по одному (таким образом, один из исполнителей останется без работы);
- каждый исполнитель может быть назначен только на одну работу;
- исполнители И2 и И5 не могут выполнять работу Р3;
- исполнителей И1 и И3 нельзя назначать на одну работу.

Для описания задачи введем переменные X_{ij}, i=1,...,5, j=1,...,3. Эти переменные должны принимать значения $X_{ij}=1$, если i-й исполнитель назначен на j-ю работу, и $X_{ij}=0$ – в противном случае.

Первое, второе и третье условия удобно сразу записать в алгебраической форме.

Для работы Р1 требуются два исполнителя, для Р2 и Р3 – по одному:

$X_{11}+X_{21}+X_{31}+X_{41}+X_{51} = 2$

$X_{12}+X_{22}+X_{32}+X_{42}+X_{52} = 1$

Библиотека БГУИР

$X_{13}+X_{23}+X_{33}+X_{43}+X_{53} = 1$

Каждый исполнитель может быть назначен только на одну работу:

$X_{11}+X_{12}+X_{13} \le 1$

$X_{21}+X_{22}+X_{23} \le 1$

$X_{31}+X_{32}+X_{33} \le 1$

$X_{41}+X_{42}+X_{43} \le 1$

$X_{51}+X_{52}+X_{53} \le 1$

Исполнители И2 и И5 не могут выполнять работу Р3:

$X_{23} = 0$

$X_{53} = 0$

Условие невозможности назначения исполнителей И1 и И3 на одну работу запишем сначала с помощью логических функций:

$X_{11} \rightarrow \overline{X}_{31}$

$X_{12} \rightarrow \overline{X}_{32}$

$X_{13} \rightarrow \overline{X}_{33}$

По правилам, приведенным в табл.4.2, перейдем к алгебраической записи:

$(1-X_{31})-X_{11} \ge 0$

$(1-X_{32})-X_{12} \ge 0$

$(1-X_{33})-X_{13} \ge 0$

Требуется найти значения переменных X_{ij}, $i=1,...,5$, $j=1,...,3$, соответствующие всем указанным условиям. Для этого используем следующую программу на Прологе.

```
predicates
nondeterm poisk
nondeterm znach (integer)

goal

poisk.
clauses
poisk:- znach(X11), znach (X12), znach (X13),
        znach(X21), znach (X22), znach (X23),
        znach(X31), znach (X32), znach (X33),
        znach(X41), znach (X42), znach (X43),
        znach(X51), znach (X52), znach (X53),
        X11+X21+X31+X41+X51 = 2,
        X12+X22+X32+X42+X52 = 1,
         X13+X23+X33+X43+X53 = 1,
         X11+X12+X13 <= 1,
         X21+X22+X23 <= 1,
```

Библиотека БГУИР

```
        X31+X32+X33 <= 1,
        X41+X42+X43 <= 1,
        X51+X52+X53 <= 1,
        X23 = 0, X53 = 0,
        (1-X31)-X11 >= 0,
        (1-X32)-X12 >= 0,
        (1-X33)-X13 >= 0,
        write("Вариант распределения: "), nl,
        write("X11=", X11, "  X12=", X12, "  X13=", X13), nl,
        write("X21=", X21, "  X22=", X22, "  X23=", X23), nl,
        write("X31=", X31, "  X32=", X32, "  X33=", X33), nl,
        write("X41=", X41, "  X42=", X42, "  X43=", X43), nl,
        write("X51=", X51, "  X52=", X52, "  X53=", X53), nl,
        readchar(_), fail.
poisk:- write("Все варианты найдены").

znach(X):- X=0.
znach(X):- X=1.
```

Работа этой программы аналогична предыдущей. Предикат fail, указанный в конце предиката poisk, требуется для того, чтобы найти все варианты решения.

Порядок выполнения работы

По заданию, выданному преподавателем, представить набор заданных утверждений с помощью логических функций. Перейти к алгебраическому представлению. Разработать и отладить программу на языке Пролог для решения полученной логической задачи.

Контрольные вопросы

1. Основные логические функции.
2. Представление логических функций в алгебраической форме.
3. Примеры представления знаний с помощью логических функций и в алгебраической форме.
4. Обработка знаний, представленных с помощью логических функций, в программах на Прологе.

ЛАБОРАТОРНАЯ РАБОТА №5

ПОСТРОЕНИЕ БАЗЫ ЗНАНИЙ ПРОДУКЦИОННОЙ ЭКСПЕРТНОЙ СИСТЕМЫ

Цель работы. Изучение возможностей представления знаний в виде продукционных правил. Реализация продукционной базы знаний средствами языка Пролог.

Библиотека БГУИР

5.1. Структура экспертной системы

Экспертная система (ЭС) - это компьютерная система, содержащая знания специалистов-экспертов в некоторой предметной области и способная принимать решения, близкие по качеству к решениям эксперта.

В структуре экспертной системы обычно выделяют следующие компоненты:

- база знаний и данных;
- механизм вывода;
- интерфейс пользователя;
- подсистема объяснения;
- подсистема извлечения знаний.

База знаний (БЗ) содержит знания, относящиеся к конкретной предметной области, в том числе факты, правила (отношения между фактами), оценки достоверности фактов и правил (например, коэффициенты уверенности). Знания экспертов в БЗ представляются в некоторой стандартной форме, например, в виде продукционных правил.

Механизм вывода - это программные средства, обеспечивающие поиск решений на основе базы знаний. Действия механизма вывода аналогичны рассуждениям человека-эксперта. Механизм вывода выполняет поиск решений в базе знаний, а также оценивает достоверность предлагаемых решений.

Интерфейс пользователя обеспечивает обмен информацией между пользователем и ЭС. Обычно интерфейс пользователя строится на основе системы меню. Кроме того, существуют интерфейсы на основе командного языка ЭС и элементов естественного языка.

Подсистема объяснения - это программные средства, объясняющие пользователю процесс вывода решения. Эта подсистема должна иметь возможность объяснять следующее: как ЭС пришла к какому-либо выводу (ответ на вопрос "как"); для чего у пользователя запрашивается та или иная информация (ответ на вопрос "почему").

Подсистема извлечения знаний предназначена для пополнения и корректировки базы знаний. В этой подсистеме могут быть реализованы методы приобретения знаний у экспертов. Такие методы могут быть прямыми (у эксперта непосредственно запрашиваются его знания) или косвенными (например, обучение ЭС по набору примеров рассуждений, приведенных экспертом). Могут применяться также методы обучения ЭС в процессе работы (например, методы обучения на ошибках).

Библиотека БГУИР

5.2. Представление знаний в продукционных ЭС

Продукционная ЭС - это экспертная система, в которой знания экспертов представлены в виде правил "если - то", т.е. продукционных правил. Каждое правило представляет собой некоторое условное утверждение (например, ЕСЛИ условие, ТО заключение; ЕСЛИ ситуация, ТО действие). Представление знаний в виде набора правил имеет следующие основные достоинства:

- естественность: человек-эксперт во многих случаях выражает свои знания именно в форме правил;
- модульность: каждое правило представляет собой один относительно независимый фрагмент знаний;
- удобство модификации базы знаний (как следствие модульности); можно добавлять новые и изменять существующие правила, не изменяя при этом других правил;
- прозрачность: удобство объяснения процесса вывода решения.

Основными элементами правила являются посылка (условие применимости правила) и заключение (действие, выполняемое в случае истинности посылки). Кроме того, в структуру правила могут входить также метка (номер правила или некоторое поясняющее обозначение), элементы для объяснения (комментарии), оценка достоверности правила и некоторые другие элементы.

5.3. Представление базы знаний средствами языка Пролог

Рассматриваемый в данной работе вариант представления базы знаний близок к форме представления знаний, используемой в оболочке для построения продукционных экспертных систем GURU.

Основными элементами продукционного правила являются посылка (условная часть) и заключение (действие).

Условная часть правила - это набор условий ("больше", "меньше", "равно" и т.д.). Будем считать, что все условия, составляющие условную часть правила, объединены логической операцией “и”. Если требуется задать условие с использованием операции “или”, то следует ввести в базу знаний несколько правил (по одному для каждого из условий, связанных операцией “или”).

Заключение правила - это действие, выполняемое при истинности его условной части. Для простоты будем считать, что в правилах возможно только одно действие: присваивание значения переменной. Будем также считать, что в каждом правиле может быть только одно действие.

При реализации базы знаний для продукционной ЭС средствами языка Пролог каждое правило будем представлять в виде предиката базы данных.

Библиотека БГУИР

Этот предикат будет содержать основные элементы правила (посылку и заключение), а также его номер. Для предикатов, описывающих правила, будем использовать имя rule (правило). Эти предикаты можно объявить следующим образом:

```
global facts
rule (integer, usl, zakl)
```

В предикате rule первый аргумент (с типом integer) - номер правила; usl - условная часть правила (посылка); zakl - заключение (действие). Очевидно, что типы usl и zakl - нестандартные (в Прологе таких типов нет). Поэтому их необходимо объявить в разделе global domains.

Условную часть правила объявим в разделе global domains следующим образом:

```
uslovie = eq (string, string);
          ne(string, string);
          gt (string, string);
          lt  (string, string);
          le (string, string);
          ge(string, string)
usl=uslovie*
```

Здесь eq, ne, gt, lt, le, ge - функторы (см. подраздел 3.1), с помощью которых будут представляться отдельные условия, составляющие условную часть. Функтором eq будем обозначать условие "равно", ne - "не равно", gt- "больше", lt - "меньше", le - "меньше или равно", ge - "больше или равно". Здесь использованы обозначения операций сравнения, образованные от соответствующих английских слов. Они не являются какими-либо зарезервированными словами, поэтому можно использовать любые другие обозначения (например, вместо eq – rav, вместо ne – nerav, и т.д.). Будем считать, что во всех этих функторах первым аргументом является имя переменной, а вторым - некоторая константа, с которой сравнивается значение переменной. Для простоты будем считать, что все величины, обрабатываемые в ЭС - строковые.

Тип uslovie - альтернативный домен. Объект этого типа может представлять собой *любой из функторов* eq, ne, gt, lt, le, ge (но только *один*). Тип usl - *список* объектов типа uslovie. Таким образом, объект с типом uslovie может представлять собой *список из любого количества функторов*. Такое объявление позволяет описывать условные части правил, состоящие из произвольного количества различных условий.

Библиотека БГУИР

Заключение правила объявим в разделе global domains следующим образом:

zakl = assign (string, string)

Здесь assign - функтор, которым будет обозначаться присваивание переменной некоторого значения (вместо assign можно использовать любое другое имя). Имя переменной будем указывать первым аргументом, а присваиваемое значение - вторым.

Кроме правил, в базе знаний будем хранить описания переменных, используемых в ЭС. Описание переменной будет состоять из двух частей: имя переменной, а также запрос, выводимый на экран при вводе значения данной переменной. Важно отметить, что запрос требуется только для переменных, значения которых не определяются самой ЭС в ходе обработки правил, а запрашиваются у пользователя. Описание каждой переменной будем представлять в виде предиката базы данных с именем var (можно использовать любое другое имя). Этот предикат необходимо объявить в разделе global facts следующим образом:

var (string, string)

Первый аргумент этого предиката будет представлять собой имя переменной, второй - текст запроса (если он требуется).

Необходимо также указать целевую переменную, т.е. переменную, значение которой необходимо определить в результате работы ЭС. Для ее указания объявим в разделе global facts следующий предикат базы данных:

goal_var (string)

Аргументом этого предиката базы данных будет имя целевой переменной.

Примечание. Следует еще раз обратить внимание, что все обозначения, использованные при описании базы знаний (rule, eq, lt, goal_var и т.д.) не являются какими-либо стандартными обозначениями языка Пролог. Вместо них можно использовать любые другие обозначения (например, вместо rule - pravilo, вместо var - perem, вместо assign - prisv, и т.д.).

Примечание. Операции, обозначенные с помощью функторов (сравнение, присваивание и т.д.), будут реализованы при построении механизма вывода.

Примечание. Не следует путать переменные, используемые в правилах (т.е. переменные ЭС), с переменными программы на языке Пролог. Как будет показано ниже, переменные ЭС для программы на языке Пролог - это просто данные, обрабатываемые программой в ходе ее выполнения.

5.4. Пример базы знаний

Пусть в ЭС, используемой для диагностики неисправностей некоторого устройства, требуется указать следующие правила:

Библиотека БГУИР

- если температура выше 200ºC, то имеется неисправность типа 1;
- если температура - от 100 до 200ºC, и при этом есть вибрация, то имеется неисправность типа 2, а при той же температуре и отсутствии вибрации - неисправность типа 3;
- при неисправности типа 1 или 2 необходимо прекратить работу устройства, при неисправности типа 3 - сменить режим работы.

База знаний, содержащая эти правила, может выглядеть так:

```
goal_var("D")
var("T","Введите температуру: ")
var("Vibr","Есть ли вибрация (да/нет): ")
var("Neispr","")
var("D","")
rule(1,[gt("T","200")],assign("Neispr","тип 1"))
rule(2,[gt("T","100"),le("T","200"),eq("Vibr","да")],assign("Neispr","тип 2"))
rule(3,[gt("T","100"),le("T","200"),eq("Vibr","нет")],assign("Neispr","тип 3"))
rule(4,[eq("Neispr","тип 1")],assign("D","прекратить работу"))
rule(5,[eq("Neispr","тип 2")],assign("D","прекратить работу"))
rule(6,[eq("Neispr","тип 3")],assign("D","сменить режим работы"))
```

Примечание. Каждое правило (как и любой факт, хранящийся в базе данных на Прологе) должно быть набрано в одну строку.

Здесь переменная D объявлена как целевая.

Для переменных Neispr и D не указан текст запроса, так как эти переменные не запрашиваются у пользователя, а определяются самой ЭС на основе правил и данных, предоставленных пользователем (температуры и вибрации). Однако указание пустой строки (“”) вместо текста запроса для этих переменных *обязательно*, так как в объявлении предиката var указано, что он имеет *два* аргумента.

5.5. Построение ЭС на языке Пролог

Приведем пример программы на языке Пролог, представляющей собой демонстрационный вариант продукционной ЭС. Программа реализует следующие действия: загрузку базы знаний ЭС, ее просмотр, сеанс работы с ЭС (консультацию). Таким образом, из основных элементов ЭС (см. подраздел 5.1) в этой программе предусмотрена реализация базы знаний и механизма вывода.

Структура и принцип работы предлагаемой программы аналогичны программе для работы с базой данных, рассмотренной в лабораторной работе 3 (подраздел 3.8).

```
goal
start
clauses
start:-clearwindow,
```

Библиотека БГУИР

```
        makewindow (1,7,7,"Экспертная система",0,0,8,60),
        repeat, clearwindow,
        write ("1 - загрузка"), nl,
        write ("2 - просмотр базы знаний"), nl,
        write ("3 - консультация "), nl,
        write ("99 - выход"), nl,
        write ("Введите номер: "), readint (N),
        obrab (N), removewindow.
obrab(1):- makewindow(2,7,7,"",8,0,3,60), zagruzka,
           removewindow, shiftwindow(1), !,fail.
obrab(2):- makewindow(3,7,7,"База знаний",12,0,10,80), prosmotr_bz,
           removewindow, shiftwindow(1), !,fail.
obrab(3):- makewindow(4,7,7,"Консультация",12,0,10,80), vyvod,
           removewindow, shiftwindow(1), !,fail.
obrab(99).
```

Предикат zagruzka реализует загрузку базы знаний, предикат prosmotr_bz - просмотр базы знаний на экране, предикат vyvod – работу механизма вывода. Реализация предиката vyvod будет рассмотрена в следующей лабораторной работе. Предикаты zagruzka и prosmotr_bz предлагается реализовать самостоятельно. Они реализуются аналогично предикатам загрузки и просмотра базы данных, рассмотренным в лабораторной работе 3.

В предлагаемой программе необходимо указать также объявление предикатов базы данных и нестандартных типов данных в разделах global facts и global domains (см. подраздел 5.3), а также объявление всех используемых предикатов в разделе predicates.

Порядок выполнения работы

По заданию, выданному преподавателем, представить набор заданных утверждений в виде продукционных правил. Реализовать полученную продукционную базу знаний в виде базы данных на Прологе. Разработать и отладить программу на языке Пролог для работы с продукционной базой знаний, как показано в подразделе 5.5. Реализовать операции загрузки и просмотра базы знаний.

Контрольные вопросы

1. Структура экспертной системы.
2. Понятие продукционной экспертной системы.
3. Структура продукционного правила.
4. Представление продукционных правил средствами языка Пролог.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №6

ПОСТРОЕНИЕ МЕХАНИЗМА ВЫВОДА В ПРОДУКЦИОННОЙ ЭКСПЕРТНОЙ СИСТЕМЕ

Цель работы. Изучение механизма обратного логического вывода в продукционных ЭС и его реализация средствами языка Пролог.

6.1. Назначение механизма вывода. Обратный логический вывод

Механизм вывода предназначен для построения заключений на основе знаний, содержащихся в базе знаний. Действия механизма вывода аналогичны рассуждениям человека-эксперта, который оценивает проблему и предлагает возможные решения. По запросу пользователя механизм вывода выполняет поиск решений в базе знаний, а также оценивает достоверность предлагаемых решений.

В данной работе рассматривается построение механизма обратного логического вывода (обратной цепочки рассуждений). Предлагаемая реализация обратного логического вывода близка к механизму вывода, используемого в оболочке для построения продукционных ЭС GURU.

В начале процесса консультации (т.е. рассмотрения набора правил) для механизма вывода указывается целевая переменная (цель). Под целью понимается некоторая переменная, значение которой механизм вывода должен определить в результате консультации. Механизм вывода анализирует правила базы знаний до тех пор, пока не установит значение целевой переменной или пока не выяснит, что найти такое значение невозможно. В первом случае ЭС сообщает о найденном решении, во втором - о невозможности нахождения решения.

При использовании обратного вывода ЭС для нахождения значения целевой переменной просматривает заключения правил, имеющихся в базе знаний. ЭС находит правило, в заключении которого может быть определено значение цели (т.е. правило, в заключении которого выполняется присваивание целевой переменной некоторого значения). Если таких правил несколько, то выбор конкретного правила зависит от реализации конкретной ЭС; обычно для рассмотрения выбирается первое правило из тех, в заключениях которых может быть определена целевая переменная. Анализируется посылка выбранного правила. Если она верна (т.е. выполняются указанные в ней условия), то правило включается (срабатывает): выполняются действия, указанные в его заключении. В результате целевая переменная получает некоторое значение. Если посылка неверна, правило не включается, и происходит обращение к

Библиотека БГУИР

следующему правилу, в заключении которого может быть определена целевая переменная. Если такого правила нет, то консультация завершается с выдачей сообщения о невозможности нахождения решения.

Если посылка выбранного правила содержит одну или несколько переменных, значения которых неизвестны системе (еще не найдены), то такие переменные поочередно (обычно - в порядке их расположения в посылке, слева направо) рассматриваются в качестве временных целей. ЭС пытается определить значение каждой из временных целей так же, как это делается для основной целевой переменной: отыскиваются правила, в которых временная цель может получить значение, проверяются посылки этих правил и т.д. Этот процесс может повторяться многократно, так как для определения временной цели в одном из правил может потребоваться найти значения других переменных, содержащихся в посылке этого правила, и т.д.

После определения значений неизвестных переменных ЭС выполняет проверку посылки правила, для которого эти переменные потребовались (т.е. правила, в заключении которого определяется целевая переменная). В зависимости от реализации механизма вывода, проверка посылки правила может производиться один раз (после определения всех неизвестных переменных) или многократно (после определения каждой из неизвестных переменных). В последнем случае поиск значений всех неизвестных переменных может и не потребоваться. Например, если условия в посылке связаны логической операцией "и", и одно из условий не выполняется, то поиск неизвестных переменных для проверки других условий не требуется: рассмотрение данного правила прекращается (так как уже определено, что оно не должно включаться), и для рассмотрения выбирается следующее правило.

6.2. Реализация обратного логического вывода

Примечание. При составлении программы на языке Пролог, реализующей логический вывод, необходимо учитывать, что все переменные, указанные в правилах базы знаний - это не то же самое, что переменные программы на Прологе. Для программы все переменные, используемые в правилах базы знаний, являются *не переменными, а данными для обработки*. Не следует также путать целевую переменную продукционной ЭС (которая является для программы просто элементом данных) и целевой предикат программы на Прологе, указываемый в разделе goal.

Для указания целевой переменной и выполнения ее поиска можно воспользоваться следующим предикатом.

```
vyvod:-
    retractall (znach(_,_)),
    goal_var (G),
    obr_vyvod (G,Result),
```

Библиотека БГУИР

```
    nl, write ("Решение: ", Result),
    readchar(_), !.
vyvod:-
    nl, write ("Цель не найдена"), readchar (_).
```

Здесь стандартный предикат retractall удаляет из памяти все предикаты базы данных znach, созданные в ходе предыдущей консультации (назначение предикатов znach будет рассмотрено ниже). Если таких предикатов нет, то предикат retractall завершается успешно, не выполняя никаких действий (см. лабораторную работу 3). Затем определяется имя целевой переменной ЭС из предиката базы данных goal_var. Это имя присваивается переменной программы G. Предикат obr_vyvod непосредственно реализует процесс обратного логического вывода. В результате его доказательства определяется значение целевой переменной (переменная Result), и оно выводится на экран. Если вывести значение целевой переменной не удается, то доказательство предиката obr_vyvod заканчивается неудачей. В этом случае заканчивается неудачей доказательство первого клоза предиката vyvod, и доказывается его второй клоз: на экран выводится сообщения о невозможности определения цели.

Предикат obr_vyvod можно реализовать следующим образом.

```
obr_vyvod (G,Result):- znach (G,Result), !.

obr_vyvod (G,Result):-
     var (G,Zapros), Zapros<>"",
     write (Zapros), readln (Result),
     assert (znach(G,Result)), !.

obr_vyvod(G,Result):-
     rule (N,Usl,assign(G,Result)),
     proverka (Usl),
     assert (znach(G,Result)), !.
```

Таким образом, для реализации предиката obr_vyvod использованы три клоза. В первом из них проверяется, нет ли в базе данных факта (предиката znach), который указывал бы значение целевой переменной.

Во втором клозе предпринимается попытка найти в базе данных описание целевой переменной, т.е. предикат var, первым аргументом которого является переменная с именем, хранящимся в программной переменной G. Затем проверяется, предусмотрен ли запрос этой переменной (имеется ли в описании переменной текст запроса). Если в базе данных удается найти соответствующий предикат var с текстом запроса, то этот текст выводится на экран (предикатом write), и у пользователя ЭС запрашивается значение переменной, которая в данный момент является целевой. Это значение сохраняется в базе данных (т.е.

Библиотека БГУИР

в памяти): для этого стандартный предикат assert создает предикат базы данных znach, аргументами которого является имя переменной и ее значение, введенное пользователем. Это требуется, чтобы не повторять запрос переменной, если затем в процессе консультации снова потребуется ее значение.

Примечание. Так как znach - предикат базы данных, его необходимо объявить в разделе global facts.

Важно отметить, что для основной целевой переменной (указанной в предикате базы знаний goal_var) обращение к первым двум клозам *всегда* заканчивается неудачей, так как в начале консультации значение целевой переменной неизвестно (еще не определено), а в базе знаний (конечно, если она правильно составлена) *не может быть* предусмотрен запрос основной целевой переменной у пользователя.

В третьем клозе отыскивается правило (предикат базы знаний rule), в котором определяется значение цели. Для такого правила проверяется условная часть с помощью предиката proverka. Если условная часть оказывается верной (предикат proverka доказывается успешно), то выполняется действие, указанное в заключении правила: переменной Result присваивается значение, заданное в функторе assign. Имя найденной целевой переменной и ее значение заносятся в базу данных в форме предиката znach. Таким образом, найденное значение переменной становится фактом базы данных и может использоваться в последующих выводах (если эта переменная не была основной целевой переменной). Предикат отсечения (!) требуется для того, чтобы при выполнении одного из правил остальные правила, в которых определяется та же цель, не сохранялись в памяти как неиспользованные альтернативы.

Если условная часть оказывается неверной (доказательство предиката proverka завершается неудачей), то происходит возврат к следующему правилу (т.е. предикату rule), в котором определяется та же целевая переменная.

Предикат proverka, выполняющий проверку условной части правила, можно реализовать следующим образом.

```
proverka([]):- !.
proverka(Usl):- Usl=[H|T],
                prov1 (H),
                proverka (T).
```

Условная часть (посылка) правила представляет собой *список* условий, каждое из которых реализовано в виде функтора. Поэтому предикат proverka выполняет поочередную проверку этих условий, начиная с первого. Принцип

Библиотека БГУИР

работы этого предиката такой же, как и у рассмотренных ранее предикатов для обработки списков (см. лабораторную работу 2). Непосредственно проверка каждого условия реализуется предикатом prov1, который должен завершаться успешно при выполнении условия, а при невыполнении - завершаться неудачей. Если какое-либо из условий в посылке оказывается ложным (т.е. доказательство предиката prov1 для него завершается неудачей), то проверка остальных условий, т.е. доказательство предиката proverka(T), не выполняется. Предикат proverka в этом случае завершается неудачей.

Предикат prov1 предназначен для проверки каждого из условий, входящих в условную часть правила. Как отмечено выше, условная часть правила реализована в виде списка и состоит из функторов, которыми обозначены различные операции сравнения (см. лабораторную работу 5). В качестве примера рассмотрим реализацию предиката prov1 для сравнения на равенство.

```
prov1(Uslovie):- Uslovie=eq(Perem,Vel), !,
                 obr_vyvod(Perem,X),
                 X=Vel.
```

Здесь переменной Uslovie присвоен один из функторов, составляющих посылку правила (этот функтор передается из предиката proverka). Сначала проверяется, обозначает ли проверяемый функтор операцию проверки на равенство (т.е. является ли он функтором eq). Если оказывается, что проверяется другой функтор (например, ge или lt), то сравнение Uslovie=eq(Perem,Vel) заканчивается неудачей, и происходит возврат к другим клозам предиката prov1, реализующим другие операции сравнения.

Если оказывается, что выполняется сравнение на равенство, то переменная Perem связывается с именем переменной, указанной в функторе eq, а переменная Vel - с величиной, с которой требуется выполнить сравнение. Переменная, указанная в функторе, становится временной целью; для нее выполняется процесс поиска значения (обратного вывода) с помощью предиката obr_vyvod. Затем значение, найденное в результате обратного вывода (X), сравнивается с указанным в функторе (Vel); если они равны, это означает, что проверяемое условие (из посылки правила) верно. Если эти значения не равны (т.е. проверяемое условие ложно), то предикат X=Vel заканчивается неудачей; вместе с ним заканчивается неудачей и предикат prov1 (т.е. проверка данного условия). Другие клозы предиката prov1 (реализующие другие операции сравнения) при этом не рассматриваются, так как они исключены из рассмотрения предикатом отсечения (!).

Библиотека БГУИР

Предикат prov1 для других операций сравнения реализуется аналогично. Следует обратить внимание, что операции сравнения "больше", "меньше", "больше или равно" и "меньше или равно" выполняются для числовых данных, а в предлагаемой реализации ЭС все данные хранятся в строковой форме. Поэтому при проверке соответствующих операций сравнения требуется преобразование типов. Ниже приводится пример клоза предиката prov1 для проверки условия "больше".

```
prov1(Uslovie):-Uslovie=gt(Perem,Vel),!,
                obr_vyvod(Perem,X),
                str_real(Vel,Nvel),
                str_real(X,Nx),
                Nx>Nvel.
```

Для преобразования типов здесь использован стандартный предикат str_real, преобразующий свой первый аргумент (строковую переменную) в вещественное число (второй аргумент).

6.3. Пример сеанса работы механизма вывода

Рассмотрим процесс консультации с ЭС, используемой для диагностики неисправностей некоторого устройства. База знаний для этой ЭС приведена в лабораторной работе 5.

В предикате vyvod определяется имя целевой переменной ЭС. Для этого в базе знаний, загруженной с диска, отыскивается предикат goal_var("D"). Переменная программы G принимает значение "D". Затем начинается доказательство предиката obr_vyvod("D",Result); переменная Result пока не имеет значения (является неизвестной).

Попытка доказательства первых двух клозов предиката obr_vyvod заканчивается неудачей, так как в базе данных еще нет предикатов znach, и нет предиката var с запросом переменной "D". При доказательстве третьего клоза предиката obr_vyvod находится правило:

```
rule(4,[eq("Neispr","тип 1")],assign("D","прекратить работу"))
```

Предикат proverka выполняет проверку условной части этого правила: [eq("Neispr","тип 1")]. В предикате prov1 переменная Neispr становится временной целью: выполняется попытка доказать предикат obr_vyvod("Neispr",X), т.е. определить значение переменной Neispr. Попытка доказать первых два клоза предиката obr_vyvod заканчивается неудачей (по тем же причинам, что и для основной целевой переменной). При доказательстве третьего клоза находится правило:

Библиотека БГУИР

rule(1,[gt("T","200")],assign("Neispr","тип 1"))

Для этого правила предикат proverka выполняет проверку условной части: [gt("T","200")]. В предикате prov1 временной целью становится переменная T. Ее значение вводится во втором клозе предиката obr_vyvod, так как эта переменная запрашивается у пользователя ЭС. Пусть введено значение "120"; при этом создается предикат базы данных znach("T","120"). Сравнение 120>200 (в предикате prov1) заканчивается неудачей. Таким образом, попытка доказать условие gt("T","200") заканчивается неудачей. Поэтому завершается неудачей предикат proverka, выполнявший проверку условной части правила 1. Происходит возврат к следующему правилу, в заключении которого имеется переменная Neispr. Это правило 2:

rule(2,[gt("T","100"),le("T","200"),eq("Vibr","да")], assign("Neispr","тип 2"))

При проверке его условной части (предикатами proverka и prov1) временной целью дважды становится переменная T и один раз - Vibr. Значение переменной T определяется в первом клозе предиката obr_vyvod, так как оно уже сохранено в базе данных. Значение переменной Vibr вводится во втором клозе предиката obr_vyvod. Пусть для переменной Vibr введено значение "да"; при этом создается предикат базы данных znach(“Vibr”,”да”). В результате условная часть правила 2 оказывается верной, и в базе данных создается предикат znach("Neispr","тип 2").

На этом заканчивается доказательство предиката obr_vyvod("Neispr",X), вызванного из предиката prov1 при проверке условия eq("Neispr","тип 1"). Переменная программы X (а для ЭС – переменная Neispr) при этом получила значение "тип 2". Сравнение "тип 2"="тип 1" завершается неудачей. Поэтому заканчиваются неудачей предикаты prov1(eq("Neispr","тип 1")) и proverka ([eq("Neispr","тип 1")]). Таким образом, заканчивается неудачей проверка условной части правила 4. Происходит переход к следующему правилу (т.е. предикату rule), в котором определяется переменная "D". Это правило 5:

rule(5,[eq("Neispr","тип 2")],assign("D","прекратить работу")).

Предикаты proverka и prov1 выполняют проверку посылки этого правила. Переменная Neispr снова становится временной целью (в предикате prov1). Ее значение определяется уже в первом клозе предиката obr_vyvod, так как оно было найдено ранее: в базе данных имеется предикат znach("Neispr","тип 2"). Сравнение "тип 2"="тип 2" выполняется успешно. Поэтому успешно доказывается предикат prov1(eq("Neispr","тип 2")). Предикат proverka([eq("Neispr", "тип 2")]), проверявший условную часть правила 5, также

Библиотека БГУИР

доказывается успешно. Таким образом, правило 5 включается. В базе данных создается предикат znach("D","прекратить работу"). На этом успешно завершается доказательство предиката obr_vyvod("D",Result). Переменная Result получила значение "прекратить работу". Это значение выводится на экран в предикате vyvod.

Порядок выполнения работы

В программе, разработанной в лабораторной работе 5, реализовать механизм обратного логического вывода. Отладить его на примере работы с базой знаний, построенной в лабораторной работе 5.

Контрольные вопросы

1. Механизм вывода в ЭС. Обратный логический вывод.
2. Реализация обратного логического вывода на языке Пролог.
3. Реализация проверки условной части продукционного правила на языке Пролог.
4. Реализация ввода данных от пользователя в процессе логического вывода.
5. Пример последовательности обратного логического вывода в экспертной системе, реализованной на языке Пролог.

ЛАБОРАТОРНАЯ РАБОТА №7

МЕТОДЫ И АЛГОРИТМЫ РАСПОЗНАВАНИЯ В ЭКСПЕРТНЫХ СИСТЕМАХ

Цель работы. Изучение методов и алгоритмов распознавания, применяемых в ЭС. Изучение методов обучения ЭС на основе алгоритмов распознавания.

7.1. Назначение методов распознавания. Решающие функции

Системы распознавания предназначены для отнесения распознаваемого объекта или явления к одному из известных классов на основе признаков данного объекта (явления). Под классом здесь понимается некоторая совокупность объектов, обладающих близкими (в чем-либо) свойствами. В качестве объекта может рассматриваться реальный физический объект (например, летательный аппарат) или некоторая ситуация (неисправность, заболевание, данные метеорологических наблюдений и т.д.).

Библиотека БГУИР

Примеры применения систем распознавания - системы технической диагностики, системы медицинской диагностики, системы технического контроля, системы для составления метеорологических прогнозов и т.д.

Во многих случаях распознаваемый объект можно представить в виде набора чисел - значений признаков этого объекта, т.е. как вектор $X = (X_1,X_2,...,X_N)$, где X_j - значение j-го признака объекта, N - количество используемых признаков.

Решающая (дискриминаторная, классифицирующая) функция - это функция от значений признаков распознаваемого объекта, по значению которой может быть принято решение об отнесении объекта к одному из известных классов (или о том, что объект не может быть отнесен ни к одному из этих классов).

Рассмотрим следующие виды решающих функций:

- решающие функции на основе минимального расстояния;
- разделяющие решающие функции (границы между классами).

7.2. Решающие функции на основе минимального расстояния

Решающие функции такого вида могут применяться, если для каждого класса можно указать объект, который может рассматриваться как наиболее характерный представитель (прототип) данного класса.

Пусть распознаваемый объект может относиться к одному из M классов. Для распознавания используется N признаков. Пусть для каждого класса известен объект-прототип со значениями признаков $P_i=(P_{i1}, P_{i2},...,P_{iN})$. Для распознавания объекта используются следующие функции:

$$D_i = 2\,(X_1 \cdot P_{i1}+X_2 \cdot P_{i2}+...+X_N \cdot P_{iN}) - (P_{i1} \cdot P_{i1}+P_{i2} \cdot P_{i2}+...+P_{iN} \cdot P_{iN}),$$

где X_j, j=1,...,N - значения признаков распознаваемого объекта.

Примечание. Обычно такая решающая функция записывается в векторной форме: $D_i=2X \cdot P_i - P_i \cdot P_i$.

Решающая функция такого вида строится для каждого класса. Таким образом, находится M таких функций. Объект относится к классу, для которого значение функции D_i *максимально*.

Смысл решающих функций такого вида следующий. Представим распознаваемый объект X и объекты - прототипы классов P_i (i=1,...,M) как точки в

Библиотека БГУИР

N-мерном пространстве (т.е. как точки, имеющие N координат). Можно доказать, что значение функции D_i тем *больше*, чем *меньше* расстояние между распознаваемым объектом X и объектом-прототипом P_i, вычисляемое по обычной формуле евклидова расстояния:

$$|X - P_i| = \sqrt{(X_1 - P_{i1})^2 + (X_2 - P_{i2})^2 + \ldots + (X_N - P_{iN})^2}\ .$$

Здесь $| X - P_i |$ - расстояние между объектами X и P_i.

Пример. Разрабатывается экспертная система для проектирования объектов складского хозяйства на промышленных предприятиях. Большинство складов строится по одному из трех типовых проектов (ТП1, ТП2, ТП3). Выбор типового проекта, по которому будет строиться склад, производится на основе анализа двух характеристик склада: грузооборота (тыс. тонн в год) и необходимого запаса единовременного хранения (тонн), т.е. количества груза, которое должно храниться на складе постоянно. Имеются сведения о девяти складах, построенных по трем указанным проектам. Эксплуатация этих складов показала, что проекты были выбраны удачно. Характеристики складов приведены в табл. 7.1.

Так как используемые в задаче признаки (грузооборот и запас единовременного хранения) имеют разную размерность и существенно различаются по порядку величин, их необходимо привести к безразмерному виду. Один из способов такого преобразования - деление всех имеющихся величин на максимальное из значений соответствующего признака. В данном примере необходимо разделить все значения грузооборота на 32 тыс. тонн, а значения запаса единовременного хранения - на 800 тонн. В результате будут получены безразмерные величины, находящиеся в диапазоне от нуля до единицы. Результаты приведения к безразмерному виду представлены в табл. 7.2.

Библиотека БГУИР

Таблица 7.1

Номер склада	Проект	Грузооборот, тыс. тонн в год	Запас единовременного хранения, тонн
1	ТП1	4	400
2	ТП1	6	500
3	ТП1	7	300
4	ТП2	14	200
5	ТП2	18	300
6	ТП2	25	350
7	ТП3	23	620
8	ТП3	32	570
9	ТП3	30	800

Таблица 7.2

Номер склада	Проект	Грузооборот (X_1)	Запас единовременного хранения (X_2)
1	ТП1	0,13	0,50
2	ТП1	0,19	0,63
3	ТП1	0,22	0,38
4	ТП2	0,44	0,25
5	ТП2	0,56	0,38
6	ТП2	0,78	0,44
7	ТП3	0,72	0,78
8	ТП3	1,00	0,71
9	ТП3	0,94	1,00

Информацию об успешно реализованных проектах складов можно использовать в ЭС, применяемой для проектирования новых складов. Такая ЭС будет работать на основе методов распознавания. В качестве классов будут рассматриваться типовые проекты ТП1, ТП2, ТП3, а в качестве распознаваемых объектов - новые (проектируемые) склады. Для проектируемого склада потребуется указать планируемые величины грузооборота и запаса единовременного хранения. Задача ЭС будет состоять в том, чтобы на основе анализа этих двух признаков определить, к какому из типовых проектов *ближе* проектируемый склад.

Составим решающие функции для выбора типового проекта на основе минимального расстояния, т.е. на основе минимального различия между проектируемым складом и некоторым типичным складом, построенным по анализируемому типовому проекту. В качестве прототипов (т.е. наиболее типичных складов для каждого из проектов) будем использовать "гипотетические" склады, характеристики которых соответствуют *средним*

Библиотека БГУИР

характеристикам складов, построенных по каждому из проектов. Например, для проекта ТП1 (или для первого класса) объект-прототип будет иметь следующие значения признаков:

P_{11} = (0,13+0,19+0,22)/3 = 0,18 (по признаку X_1);

P_{12} = (0,5+0,63+0,38)/3 = 0,50 (по признаку X_2).

Таким образом, P1 = (0,18; 0,50). Аналогично найдем признаки объектов-прототипов для второго и третьего классов: P_2 = (0,59; 0,36), P_3 = (0,89; 0,83).

Составим решающие функции для определения наиболее подходящего проекта на основе минимального расстояния:

$D_1 = 2\ (X_1 \cdot 0{,}18+X_2 \cdot 0{,}5)-(0{,}18 \cdot 0{,}18+0{,}5 \cdot 0{,}5) = 0{,}36 \cdot X_1+1 \cdot X_2-0{,}28;$

$D_2 = 2\ (X_1 \cdot 0{,}59+X_2 \cdot 0{,}36)-(0{,}59 \cdot 0{,}59+0{,}36 \cdot 0{,}36) = 1{,}18 \cdot X_1+0{,}72 \cdot X_2-0{,}48;$

$D_3 = 2\ (X_1 \cdot 0{,}89+X_2 \cdot 0{,}83)-(0{,}89 \cdot 0{,}89+0{,}83 \cdot 0{,}83) =1{,}78 \cdot X_1+1{,}66 \cdot X_2-1{,}48.$

Пусть, например, требуется выбрать наиболее подходящий типовой проект для проектируемого склада. Предполагается, что грузооборот склада будет составлять примерно 12 тыс. тонн в год, а запас единовременного хранения – 200 тонн.

Проектируемый склад рассматривается как распознаваемый объект. Требуется определить, к какому классу (типовому проекту) относится этот объект. Перейдем к безразмерным значениям признаков объекта: X_1=12/32=0,38, X_2=200/800=0,25.

Для распознавания воспользуемся решающими функциями:

$D_1 = 0{,}36 \cdot 0{,}38 + 1 \cdot 0{,}25 - 0{,}28 = 0{,}11;$

$D_2 = 1{,}18 \cdot 0{,}38 + 0{,}72 \cdot 0{,}25 - 0{,}48 = 0{,}15;$

$D_3 = 1{,}78 \cdot 0{,}38 + 1{,}66 \cdot 0{,}25 - 1{,}48 = -0.39.$

Таким образом, распознаваемый объект отнесен ко второму классу. В данном случае это значит, что для проектируемого склада следует выбрать типовой проект ТП2.

7.3. Разделяющие решающие функции

Под разделяющими решающими функциями будем понимать решающие функции, представляющие собой уравнения границ, разделяющих классы.

Если распознавание объектов выполняется на основе двух признаков (т.е. распознаваемые объекты могут рассматриваться как точки на плоскости), то решающая функция представляет собой уравнение линии на плоскости (в

Библиотека БГУИР

простейшем случае - уравнение прямой). Если используются три признака, то решающая функция - это уравнение некоторой поверхности в трехмерном пространстве и т.д.

Методы построения таких функций лучше всего разработаны для случаев, когда имеются только два класса. В таких случаях строится одна решающая функция, принимающая положительные значения при подстановке в нее значений признаков объектов из одного класса, и отрицательные - для другого класса.

7.3.1. Построение разделяющих решающих функций (обучение экспертной системы)

Большинство методов построения разделяющих решающих функций основано на использовании *обучающего множества*, т.е. набора объектов, для которых уже *известно*, к каким классам они относятся. В обучающем множестве должны быть объекты из всех имеющихся классов. Построение решающей функции состоит в подборе такого вида функции, при котором она правильно распознает объекты из обучающего множества.

Общий принцип построения разделяющих решающих функций (для двух классов) следующий. Один из классов (любой) обозначается как первый; для объектов из этого класса решающая функция должна принимать положительные значения, для объектов из другого класса - отрицательные. Выбирается некоторый первоначальный вид решающей функции. В нее подставляются объекты из обучающего множества. Если решающая функция правильно распознает объект (т.е. принимает положительное значение для объекта из первого класса, или отрицательное - для объекта из второго класса), то она не изменяется. В случае неправильного распознавания объекта из первого класса (т.е. при *отрицательном или нулевом* значении функции для такого объекта) функция корректируется таким образом, чтобы ее значение *увеличилось*. В случае ошибки для объекта из второго класса (т.е. при *положительном или нулевом* значении функции для такого объекта) функция корректируется таким образом, чтобы ее значение *уменьшилось*. Процесс завершается, когда решающая функция правильно распознает *все* объекты из обучающего множества.

Рассмотрим один из алгоритмов построения решающих функций - алгоритм восприятия. Воспользуемся им для построения линейной решающей функции между двумя классами (первым и вторым): $D_{12} = W_1 \cdot X_1 + W_2 \cdot X_2 + ... + W_N \cdot X_N + W_0$, где $X_1, X_2, ..., X_N$ - признаки распознаваемого объекта; W_0, W_1,

Библиотека БГУИР

$W_2,...,W_N$ - коэффициенты решающей функции (их требуется определить). Алгоритм реализуется в следующем порядке.

1. Выбираются некоторые начальные значения коэффициентов решающей функции. Пусть, например, эти коэффициенты принимаются равными нулю: $W_0=W_1=W_2=...=W_N=0$.

2. Счетчик правильно распознанных объектов принимается равным нулю: E=0.

3. Из обучающего множества выбирается первый объект.

4. Выбранный объект подставляется в решающую функцию. В зависимости от значения решающей функции выполняются следующие действия:

- если объект распознан правильно (т.е. $D_{12}>0$ для объекта первого класса, или $D_{12}<0$ для объекта второго класса), то решающая функция не изменяется. Счетчик правильно распознанных объектов увеличивается на единицу (E=E+1);

- если неправильно распознан объект из первого класса (т.е. для такого объекта $D_{12}\leq0$), то функция корректируется в направлении увеличения. Для этого к ее коэффициентам W_1, $W_2,...,W_N$ *прибавляются* значения признаков распознаваемого объекта X_1, $X_2,...,X_N$, а коэффициент W_0 *увеличивается* на единицу. Счетчик правильно распознанных объектов *принимается равным нулю* (E=0);

- если неправильно распознан объект из второго класса (т.е. для такого объекта $D_{12}\geq0$), то функция корректируется в направлении уменьшения. Для этого из коэффициентов W_1, $W_2,...,W_N$ *вычитаются* значения признаков распознаваемого объекта X_1, $X_2,...,X_N$, а коэффициент W_0 *уменьшается* на единицу. Счетчик правильно распознанных объектов *принимается равным нулю* (E=0).

5. Если счетчик правильно распознанных объектов *равен* общему количеству объектов в обучающем множестве, то алгоритм завершается. Решающая функция построена.

6. Если счетчик правильно распознанных объектов *меньше* общего количества объектов в обучающем множестве, то выбирается очередной объект (если все объекты в обучающем множестве уже рассмотрены, то снова выбирается первый объект). Выполняется возврат на шаг 4.

Библиотека БГУИР

Таким образом, алгоритм завершается, когда решающая функция правильно распознает *все* объекты в обучающем множестве. Другими словами, построение решающей функции заканчивается после того, как в нее подставляются все объекты обучающего множества, и ни разу не требуется ее корректировка.

Рассмотрим построение решающей функции для первого и второго классов (типовые проекты ТП1 и ТП2) в приведенном выше примере.

Значения коэффициентов решающей функции считаем сначала равными нулю: $D_{12}=0 \cdot X_1+0 \cdot X_2+0$. Выбираем первый объект из обучающего множества (0,13; 0,5) и подставляем его признаки в D_{12}. Очевидно, что решающая функция равна нулю. Так как рассматривался объект из первого класса, и для него решающая функция должна быть положительной, выполняем коррекцию решающей функции, как показано выше (шаг 4 алгоритма). Функция принимает вид: $D_{12}=0{,}13 \cdot X_1+0{,}5 \cdot X_2+1$. Так как проверенный объект был распознан неправильно, счетчик правильно распознанных объектов равен нулю: E=0.

Выбираем очередной объект: X=(0,19; 0,63). Подставляем его в решающую функцию: $D_{12}=0{,}13 \cdot 0{,}19+0{,}5 \cdot 0{,}63+1 = 1{,}34$. Таким образом, $D_{12}>0$ для объекта из первого класса. Значит, объект распознан правильно и коррекция функции не требуется. Счетчик увеличивается на единицу: E=1.

Выбираем очередной объект: X=(0,22; 0,38). Подставляем его в решающую функцию: $D_{12}=0{,}13 \cdot 0{,}22+0{,}5 \cdot 0{,}38+1 = 1{,}22 > 0$. Объект распознан правильно, коррекция функции не требуется. Счетчик увеличивается на единицу: E=2.

Выбираем очередной объект: X=(0,44; 0,25). Подставляем его в решающую функцию: $D_{12}=0{,}13 \cdot 0{,}44+0{,}5 \cdot 0{,}25+1 = 1{,}18 > 0$. Объект распознан неправильно: решающая функция должна быть отрицательной, так как объект относится ко второму классу. Выполняется коррекция функции, как показано в алгоритме (шаг 4). Функция принимает вид: $D_{12}=-0{,}31 \cdot X_1+0{,}25 \cdot X_2$. Счетчик принимается равным нулю (E=0). Это означает, что решающая функция изменилась, и ее требуется проверять для всех объектов заново.

Выбираем очередной объект: X=(0,56; 0,38). $D_{12}=-0{,}31 \cdot 0{,}56 + 0{,}25 \cdot 0{,}38 =$ $= -0{,}08 < 0$. Объект распознан правильно, коррекция не требуется, E=1.

Дальнейший ход построения решающей функции показан в табл.7.3.

Библиотека БГУИР

Таблица 7.3

Объект	Класс объекта	Значение D_{12}	Результат распознавания	Счетчик
(0,78; 0,44)	2	$-0,13 < 0$	верно	2
(0,13; 0,50)	1	$0,08 > 0$	верно	3
(0,19; 0,63)	1	$0,1 > 0$	верно	4
(0,22; 0,38)	1	$0,03 > 0$	верно	5
(0,44; 0,25)	2	$-0,07 > 0$	верно	6

Таким образом, решающая функция проверена на всех объектах обучающего множества, и все объекты оказались распознанными правильно. Решающая функция имеет следующий вид: $D_{12}=-0,31 \cdot X_1+0,25 \cdot X_2$.

Если имеются только два класса, то построенная решающая функция может применяться для распознавания объектов. Распознавание выполняется подстановкой значений признаков объекта в функцию. Если значение функции оказывается положительным, то объект относится к первому классу, если отрицательным - ко второму.

Геометрическая интерпретация данного метода распознавания следующая. Пусть объекты рассматриваются как точки в N-мерном пространстве, где N - количество признаков объектов. Решающая функция представляет собой границу между классами; это значит, что объекты одного класса находятся "с одной стороны" от решающей функции, а объекты другого класса - с другой стороны. При этом функция построена таким образом, что для объектов первого класса она принимает положительные значения, а для второго - отрицательные. Таким образом, если при подстановке признаков некоторого объекта функция принимает положительное значение, это значит, что объект находится "с той же стороны" от решающей функции, что и объекты первого класса, входившие в обучающее множество.

Воспользуемся построенной решающей функцией, чтобы выбрать типовой проект для склада с грузооборотом 12 тыс. тонн в год и запасом единовременного хранения 200 тонн. Безразмерные значения признаков объекта: $X_1=0,38$; $X_2=0,25$ (см. пример в подразделе 7.2). Найдем значение решающей функции: $D_{12}= -0,31 \cdot 0,38+0,25 \cdot 0,25=-0,06$. Таким образом, если бы использовались только два типовых проекта, то для склада был бы выбран проект ТП2. Однако в данной задаче делать вывод о выборе типового проекта пока нельзя, так как при построении решающей функции не учитывался

Библиотека БГУИР

типовой проект ТП3 (третий класс). В то же время уже можно утверждать, что для склада не будет использован типовой проект ТП1.

Примечание. Построение линейных решающих функций возможно не всегда. В некоторых случаях значения признаков объектов таковы, что классы оказываются линейно-неразделимыми. На рис.7.1 классы 1 и 2, 1 и 3 являются линейно-разделимыми, а классы 2 и 3 - неразделимыми. В этих случаях применяются нелинейные решающие функции.

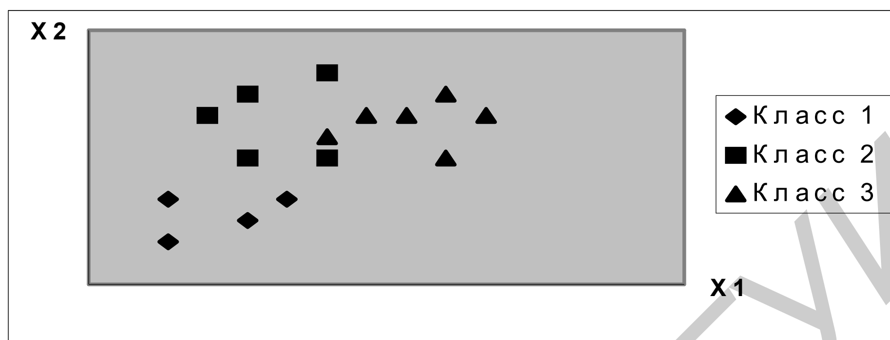

Рис.7.1. Пример линейно-неразделимых классов

7.3.2. Построение разделяющих решающих функций для нескольких классов

Имеется несколько методов построения таких решающих функций. Один из них состоит в том, что строятся разделяющие функции *для каждой пары* классов, как показано выше. В рассматриваемом примере эти функции будут иметь следующий вид:

$D_{12} = -0,31 \cdot X_1 + 0,25 \cdot X_2$ (построена выше);

$D_{13} = -1,64 \cdot X_1 - 0,34 \cdot X_2 + 1$;

$D_{23} = -0,4 \cdot X_1 - 1,34 \cdot X_2 + 1$.

Распознавание объекта выполняется на основе подстановки признаков объекта во все решающие функции. Объект относится к классу с номером k, если выполняется условие:

$D_{ki} > 0, \quad i=1,...,M, \ i \neq M, \qquad (*)$

где i - номера классов.

Геометрическая интерпретация данного способа распознавания следующая: объект относится к классу k, если он находится "со стороны этого класса" относительно всех границ класса (т.е. всех функций D_{ki}).

Воспользуемся построенными решающими функциями для рассматриваемого примера. Найдем значения решающих функций для склада с грузооборотом 12 тыс. тонн в год и запасом единовременного хранения 200

Библиотека БГУИР

тонн (безразмерные значения признаков X_1=0,38, X_2=0,25): D_{12}=-0,06; D_{13}=0,29; D_{23}=0,51. Таким образом, для склада выбирается типовой проект ТП2, так как выполняются условия $D_{21}>0$ (то же самое, что $D_{12}<0$) и $D_{23}>0$.

Если условие (*) не выполняется ни для одного из классов, то распознавание объекта невозможно. Это может означать, что распознаваемый объект сильно отличается от всех классов или, наоборот, близок сразу к нескольким классам.

Пусть, например, для некоторого склада решающие функции приняли следующие значения: D_{12}=0,17; D_{13}=-0,08; D_{23}=0,14. Для такого склада выбор проекта по построенным решающим функциям невозможен: склад не может быть отнесен ни к первому проекту (т.к. $D_{13}<0$), ни ко второму (т.к. $D_{21}<0$), ни к третьему (т.к. $D_{32}<0$). В этом случае следует использовать другие методы распознавания (например, на основе минимального расстояния).

Порядок выполнения работы

1. Получить задание на лабораторную работу. Задание должно содержать примеры объектов, относящихся к нескольким классам (обучающее множество), а также значения признаков объекта, подлежащего распознаванию.
2. Составить решающие функции на основе минимального расстояния. Выполнить распознавание заданного объекта.
3. Составить разделяющие решающие функции на основе алгоритма восприятия. Для одной пары классов (по указанию преподавателя) построить решающую функцию вручную, для остальных – с помощью программы. Выполнить распознавание заданного объекта. Построить диаграмму для геометрической интерпретации распознавания. На диаграмме должны быть показаны объекты обучающего множества, графики решающих функций и области принадлежности к каждому из классов.

Контрольные вопросы

9. Понятие решающей функции.
10. Решающие функции на основе минимального расстояния.
11. Разделяющие решающие функции. Алгоритм восприятия.
12. Обучение ЭС. Построение разделяющих решающих функций для двух и для нескольких классов.
13. Геометрическая интерпретация распознавания на основе алгоритма восприятия.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №8

БАЙЕСОВСКАЯ СТРАТЕГИЯ ОЦЕНКИ ВЫВОДОВ

Цель работы. Изучение вероятностных методов оценки достоверности выводов в ЭС.

8.1. Назначение, возможности и принцип работы байесовской стратегии оценки выводов

Байесовская стратегия оценки выводов - одна из стратегий, применяемых для оценки достоверности выводов (например, заключений продукционных правил) в ЭС. Основная идея байесовской стратегии заключается в оценке вероятности некоторого вывода с учетом фактов, подтверждающих или опровергающих этот вывод.

Формулировка теоремы Байеса, известная из теории вероятностей, следующая.

Пусть имеется n несовместных событий H_1, H_2,...,H_n. Несовместность событий означает, что *никакие* из событий H_1, H_2,...,H_n *не могут произойти вместе* (другими словами, вероятности их совместного наступления равны нулю). Известны вероятности этих событий: $P(H_1)$, $P(H_2)$,...,$P(H_n)$, причем $P(H_1)+P(H_2)+...+P(H_n)=1$; это означает, что события H_1, H_2,...,H_n образуют *полную группу* событий, т.е. *одно* из них *происходит обязательно*. С событиями H_1, H_2,...,H_n связано некоторое событие E. Известны вероятности события E *при условиях* того, что какое-либо из событий H_1, H_2,...,H_n произошло: $P(E/H_1)$, $P(E/H_2)$,..., $P(E/H_n)$. Пусть известно, что событие E произошло. Тогда вероятность того, что какое-либо из событий H_i (i=1,...,n) произошло, можно найти по следующей формуле (формула Байеса):

$$P(H_i/E) = \frac{P(E/H_i)\,P(H_i)}{P(E/H_1)P(H_1) + P(E/H_2)\,P(H_2) + ... + P(E/H_n)\,P(H_n)} = \frac{P(EH_i)}{P(E)}.$$

События H_1, H_2,...,H_n называются *гипотезами*, а событие E - *свидетельством*. Вероятности гипотез $P(H_i)$ без учета свидетельства (т.е. без учета того, произошло событие E или нет) называются доопытными (априорными), а вероятности $P(H_i/E)$ - послеопытными (апостериорными). Величина $P(EH_i)$ - *совместная* вероятность событий E и H_i, т.е. вероятность того, что произойдут

оба события вместе. Величина P(E) - полная (безусловная) вероятность события E.

Формула Байеса позволяет уточнять вероятность гипотез с учетом новой информации, т.е. данных о событиях (свидетельствах), подтверждающих или опровергающих гипотезу.

В ЭС формула Байеса может применяться для оценки вероятностей заключений продукционных правил на основе данных о достоверности их посылок. Заключения (выводы) в этом случае соответствуют гипотезам в теореме Байеса, а посылки - свидетельствам. Обычно посылка правила в ЭС содержит несколько условий. Вероятности $P(H_i)$ и $P(E/H_i)$ определяются на основе статистических данных с использованием формул теории вероятностей. Основные из этих формул следующие.

Формула умножения вероятностей (вероятность того, что произойдет и событие A, и событие B):

$$P(AB) = P(A)P(B/A) = P(B)P(A/B),$$

где P(A), P(B) - вероятности событий A и B;

P(B/A) - условная вероятность события B, т.е. вероятность события B при условии, что произошло событие A;

P(A/B) - условная вероятность события A, т.е. вероятность события A при условии, что произошло событие B.

Если события A и B *независимы* (т.е. вероятность одного события не зависит от того, произошло ли другое событие), то формула умножения вероятностей записывается следующим образом:

$$P(AB) = P(A)P(B).$$

Формула умножения вероятностей для нескольких событий (вероятность того, что произойдут *все* указанные события вместе):

$$P(A_1A_2 \ldots A_n) = P(A_1)\, P(A_2/A_1)\, P(A_3/A_1,A_2) \ldots P(A_n/A_1,A_2,\ldots,A_{n-1}).$$

Формула сложения вероятностей (вероятность того, что произойдет *хотя бы одно* из событий):

$$P(A+B) = P(A) + P(B) - P(AB).$$

Если события A и B *несовместны* (т.е. не могут произойти вместе), то P(AB)=0, и формула сложения вероятностей принимает следующий вид:

$$P(A+B) = P(A) + P(B).$$

Формула сложения вероятностей для нескольких событий обычно записывается следующим образом:

Библиотека БГУИР

$$P(A_1 + A_2 + ... + A_n) = 1 - P(\overline{A}_1 + \overline{A}_2 + ... + \overline{A}_n),$$

где $P(\overline{A}_1 + \overline{A}_2 + ... + \overline{A}_n)$- вероятность того, что не произойдет ни одного из событий A_1, A_2,...A_n. Эту величину можно найти, например, по формуле умножения вероятностей.

8.2. Пример применения байесовской стратегии оценки выводов

Пример. В ЭС для составления прогнозов погоды вероятность дождя на следующий день определяется с учетом трех факторов: ветер, влажность, облачность (в день наблюдения). За 173 дня (из них 53 дождливых) накоплены статистические данные, приведенные в табл.8.1.

Таблица 8.1

Погода в день наблюдения		Количество случаев дождливой погоды на следующий день	Количество случаев погоды без осадков на следующий день
Ветер	Слабый	19	52
	Умеренный	27	44
	Сильный	7	24
Влажность	Высокая	35	18
	Средняя	12	42
	Низкая	6	60
Облачность	Ясно	5	83
	Облачно	8	27
	Пасмурно	40	10

Приведенные в таблице данные означают, например, следующее: за период наблюдений (173 дня) слабый ветер наблюдался 71 день (71=19+52). В 19 случаях на следующий день погода была дождливой, в 52 случаях – без осадков.

В некоторый день наблюдается следующая погода: сильный ветер, высокая влажность, облачно. Требуется найти вероятность дождливой погоды на следующий день.

В данном случае в качестве гипотез рассматриваются состояния погоды на следующий день: H_1 - дождь, H_2 - погода без осадков. Свидетельством здесь является сочетание трех факторов, характеризующих погоду в день наблюдения: ветер, влажность и облачность (можно сказать, что в данном случае используются три свидетельства); обозначим эти факторы как E_1, E_2, E_3. Обозначим наблюдаемое сочетание факторов (сильный ветер, высокая влажность, облачно) как событие E.

Библиотека БГУИР

Определим вероятности, необходимые для расчетов по формуле Байеса. Априорные вероятности гипотез (т.е. вероятности дождя и погоды без осадков без учета наблюдаемого состояния погоды):

$P(H_1)=53/173=0,306;$

$P(H_2)=120/173=0,694.$

Наблюдаемое свидетельство (состояние погоды в день наблюдения) представляет собой сочетание трех событий, наблюдаемых вместе: сильного ветра, высокой влажности и облачности. Считая эти события независимыми (т.е. считая, например, что влажность не зависит от облачности, и т.д.), можно найти условные вероятности *свидетельства* по формуле умножения вероятностей:

$$P(E/H_i) = P(E_1,E_2,E_3/H_i) = P(E_1/H_i)\,P(E_2/H_i)\,P(E_3/H_i), \quad i=1,2.$$

Найдем величины, необходимые для применения формулы умножения вероятностей:

$P(E_1/H_1)=7/53=0,132;$ $P(E_2/H_1)=35/53=0,66;$ $(E_3/H_1)=8/53=0,151;$

$P(E_1/H_2)=24/120=0,2;$ $P(E_2/H_2)=18/120=0,15;$ $P(E_3/H_2)=27/120=0,225.$

Здесь, например, $P(E_1/H_1)$ - вероятность того, что *в текущий* день наблюдается сильный ветер, при условии, что *следующий* день будет дождливым. Эта величина показывает, насколько часто наблюдается сильный ветер в дни, предшествующие дождливой погоде.

Подставляя найденные величины в формулу умножения вероятностей, получим:

$P(E/H_1) = 0,132\cdot 0,66\cdot 0,151 = 0,0132;$

$P(E/H_2) = 0,2\cdot 0,15\cdot 0,225 = 0,00675.$

Здесь, например, величина $P(E/H_1)$ - вероятность наблюдаемого состояния погоды (сильного ветра, высокой влажности и облачности) при условии, что *следующий* день будет дождливым.

Найдем вероятность дождливой погоды на следующий день при наблюдаемом состоянии погоды (апостериорную вероятность):

$$P(H_1/E) = \frac{P(E/H_1)P(H_1)}{P(E/H_1)P(H_1) + P(E/H_2)P(H_2)} = \frac{0,0132\cdot 0,306}{0,0132\cdot 0,306 + 0,00675\cdot 0,694} = 0,463.$$

Библиотека БГУИР

Эта величина является более точной оценкой вероятности дождя на следующий день, чем априорная вероятность $P(H_1)$, рассчитанная на основе статистических данных без учета погоды в текущий день.

Следует также отметить, что полученная апостериорная вероятность (0,463) больше, чем априорная (0,306). Это означает, что наблюдаемые свидетельства (сильный ветер, высокая влажность, облачность) *подтверждают* гипотезу о том, что погода на следующий день будет дождливой.

Примечание. Байесовская стратегия вывода может также применяться в ЭС, в которых для оценки достоверности выводов применяются не вероятности, рассчитанные по статистическим данным, а субъективные оценки достоверности (коэффициенты уверенности), указанные экспертами.

Порядок выполнения работы

По заданию, выданному преподавателем, рассчитать вероятность указанной гипотезы на основе байесовской стратегии оценки.

Контрольные вопросы

1. Назначение байесовской стратегии оценки выводов.
2. Теорема Байеса.
3. Пример оценки достоверности гипотезы на основе байесовской стратегии (на основе задания, выполненного в ходе лабораторной работы).
4. Интерпретация вероятностей, получаемых в ходе расчетов на основе байесовской стратегии.

Библиотека БГУИР

ЛИТЕРАТУРА

1. Гаврилова Т.А., Червинский В.Ф. Базы знаний интеллектуальных систем. СПб.: Питер, 2000. – 384 с.
2. Герман О.В. Введение в теорию экспертных систем и обработку знаний. Мн.: ДизайнПРО, 1995. – 255 с.
3. Джексон П. Введение в экспертные системы. М.: Издательский дом “Вильямс”, 2001. - 622 с.
4. Железко Б.А., Морозевич А.Н. Теория и практика построения информационно-аналитических систем поддержки принятия решений. Мн.: Армита-Маркетинг, Менеджмент, 1999. - 143 с.
5. Кокорева Л.В., Перевозчикова О.Л., Ющенко Е.Л. Диалоговые системы и представление знаний: Справ. пособие. Киев: Наукова думка, 1993. – 448 с.
6. Таунсенд К., Фохт Д. Проектирование и программная реализация экспертных систем на персональных ЭВМ. М.: Финансы и статистика, 1990. – 320 с.
7. Экспертные системы для персональных компьютеров: методы, средства, реализации: Справ. пособие. / В.С. Крисевич, Л.А. Кузьмич, А.М. Шиф и др. – Мн.: Выш.шк., 1990. – 197 с.

Библиотека БГУИР