Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение «Томский государственный университет систем управления и радиоэлектроники»

(ТУСУР)

УТВЕРЖДАЮ

Заведующий кафедрой ЭМИС

_____ И.Г. Боровской

«____» _____ 2010 г.

ПОСОБИЕ К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

«Интеллектуальные информационные системы»

Составлено кафедрой «экономической математики, информатики и статистики»

Для студентов, обучающихся по направлению подготовки 230200 – Информационные системы

Форма обучения – очная

Составитель:

Должность, место работы _____ А.А. Матолыгин

«____» _____ 2010 г.

Томск 2010 г.

Методические указания к самостоятельной работе студентов составлены в соответствии с требованиями государственного образовательного стандарта высшего профессионального образования по направлению подготовки 230200 – Информационные системы.

.

Методические указания направлены на приобретение студентами необходимых навыков программирования на языке Пролог.

ОГЛАВЛЕНИЕ

Лабораторная работа 1 4
Лабораторная работа 2 7
Лабораторная работа 3 11
Лабораторная работа 4 13
КОНТРОЛЬНЫЕ ВОПРОСЫ 20
ЛИТЕРАТУРА 20

Лабораторная работа 1

Введение в язык ПРОЛОГ. Простейшие программы

Цель работы: Знакомство с синтаксисом и получение практических навыков составления простейших программ на языке логического программирования – ПРОЛОГ.

Программирование в логических языках состоит в описании модели рассматриваемой предметной области. Модель строится в терминах объектов и отношений. Отношения между объектами могут быть заданы в виде фактов и правил. *Факт* – это некоторое утверждение, истинность которого считается доказанной.

Простейшая ПРОЛОГ-программа состоит из одного факта и имеет следующий вид:

A.

что интерпретируется следующим образом:

«утверждение ***A*** выполнимо», «факт ***A*** имеет место», «***A*** истинно».

Обращение к ПРОЛОГ-программе организовано с помощью *вопросов*. Пример обращения: **? - *R***.

Интерпретация этого предложения такова:

«выполнимо ли утверждение ***R*** ?», «имеет ли место факт ***R*** ?», «истинно ли ***R*** ?»

В ПРОЛОГе принята следующая концепция: «отсутствие факта означает его неудачное выполнение», поэтому, если программа состоит из одного факта «***A.***», а вопрос задан «***?- B.***», то ответом будет «нет» («неудача», «ложь», «No»), если же задан вопрос «***?- A***.», то ответом будет «да» («удача», «истина», «Yes»).

Синтаксически вопрос от факта отличается наличием вначале предложения конструкции «***?-*** ». Вопрос и факт обязательно заканчиваются точкой « . ». Утверждение без точки будем называть *целью*.

Следующий уровень сложности ПРОЛОГ-программы связан с рассмотрением новой конструкции языка, называемой *правилом.* Правило имеет следующий вид:

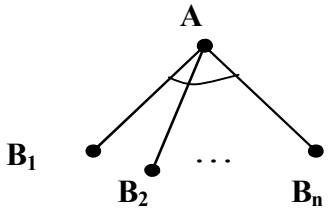
$$\mathbf{A:- B_1, B_2, \dots , B_n.} \qquad (1)$$

Состоит правило из двух частей, разделенных конструкцией «**:-** », которую можно читать как «если». Левая часть правила («***A***») называется заголовком (головой предложения) и является атомарной формулой. Правая часть правила называется телом правила и представляет собой конечное

множество целей, возможно пустое, здесь $B_1, B_2, \ldots, B_n$ – факты либо правила, соединенные в единую конструкцию символами « , ». Таким образом, символ « , » означает в теле правила связку «и» (конъюнкцию). Интерпретируется правило следующим образом:

«утверждение A выполнимо, если выполнимы все $B_1, B_2, \ldots, B_n$», «заключение A имеет место, если имеют место одновременно и B_1 *и* $B_2, \ldots,$ *и* B_n», «A истинно, если истинны все $B_1, B_2, \ldots, B_n$».

Графически правило (1) может быть представлено вершиной типа И:

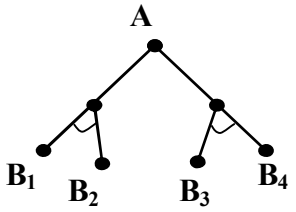


Факт можно рассматривать как частный случай правила, когда тело правила пусто ($n = 0$).

Используя понятие *процедуры*, ПРОЛОГ-программу, имеющую вид:

$$\mathbf{A{:}\text{-}\ B_1, B_2.}$$
$$\mathbf{A{:}\text{-}\ B_3, B_4.} \qquad (2)$$

можно прочитать следующим образом: если при выполнении процедуры A успешно выполнены процедуры B_1 и B_2 , то процедура A выполнена успешно; если хотя бы одна из процедур B_1 или B_2 завершилась неуспешно, то выполняется второе предложение процедуры A, и соответственно выполняются процедуры B_3 и B_4; если обе эти процедуры закончены успешно, то процедура A успешно завершена, в противном случае выполнение процедуры A завершается неуспехом. Таким образом, правила в процедуре связаны связкой «или» (дизъюнкция). Графически правила (2) могут быть представлены в виде дерева с вершиной типа ИЛИ и вершинами типа И:



В прологе принята стратегия поиска решения в глубину, когда ведется последовательный обход дерева И-ИЛИ сверху вниз и слева направо

Пример 1

Пусть имеются утверждения: «Овладеть языком программирования можно (PL), если работать с ним практически (W) и выучить его основы (F). Работа с языком практически возможна, если есть персональный компьютер

(P). Выучить основы языка можно при условии работы в библиотеке (L). Выполнены условия работы в библиотеке и наличия персонального компьютера. Следовательно, можно овладеть языком программирования». ПРОЛОГ-программа задающая, описания приведенных выше утверждений, имеет следующий вид:

PL:- W, F.
W:- P.
F:- L.
L.
P.

Обращение к программе: **?- PL.,** даст ответ «да».

Допустим теперь, что условие работы в библиотеке выполнено, а персонального компьютера нет в наличии. ПРОЛОГ-программа в этом случае будет иметь следующий вид:

PL:- W, F.
W:- P.
F:- L.
L.
P:- fail.

Здесь автором пособия использован встроенный системный предикат ***fail***, который всегда приводит к безуспешному выполнению правила, в котором ***fail*** использован. Более привычная и наглядная запись отрицания в виде ***not(P)*** невозможна в силу языковых ограничений. Вопрос к программе: **?- PL.** даст ответ «нет».

Пример2

Халиф Омар, сжегший Александрийскую библиотеку, рассуждал так: «если ваши книги согласны с Кораном (A), то они излишни (E); если ваши книги не согласны с Кораном (A), то они вредны (H); но вредные или излишние книги следует уничтожать (D); значит, ваши книги следует уничтожить».

На языке логики предикатов данные рассуждения можно представить следующим образом:

$$\frac{A \to E, \quad \neg A \to H, \quad (E \vee H) \to D \quad A \vee \neg A}{D}$$

ПРОЛОГ-программа будет иметь следующий вид:

E:- A.
H:- not(A).
D:- H.

D:- E.

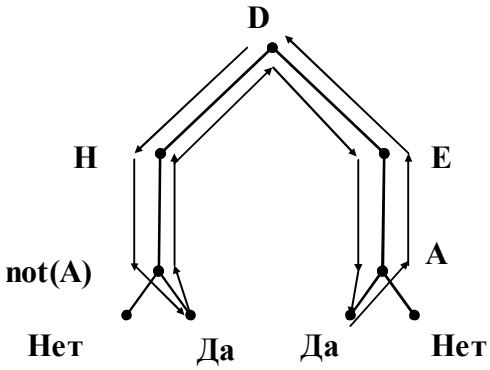
A.

A:- fail.

На вопрос о том, уничтожать ли Александрийскую библиотеку:

?- D., следует ответ **Да**.

На графе И-ИЛИ программа и поиск решения представляются следующим образом:



Исходными элементами описания предметной области в приведенных выше примерах являются отдельные утверждения: простые – факты и сложные – правила. При этом не рассматривались состав и структура утверждения в смысле связи действия, агента, объекта и других грамматических составляющих предложения.

Задание

1. Создайте предикат, вычисляющий сумму цифр натурального числа.

2. Создайте предикат, вычисляющий произведение цифр натурального числа.

3. Создайте предикат, переводящий число из десятичной системы счисления в двоичную.

4. Создайте предикат, переводящий число из одной системы счисления в другую.

Лабораторная работа 2

Базы данных

Цель работы: получение практических навыков составления программ на языке логического программирования – ПРОЛОГ, обрабатывающие базы данных.

Для углубленного анализа и описания предметной области язык программирования должен иметь возможность работы с константами,

переменными и сложными структурами. ПРОЛОГ обладает такими возможностями.

Рассмотрим следующий факт:

«Том родитель Боба».

Сказуемым или предикатом в данном предложении является «родитель», этот предикат связывает два объекта «Том» и «Боб». В языках логического программирования для записи подобных фактов существует стандартная форма записи:

предикат(объект_1, объект_2).

Тогда наш факт может быть записан следующим образом:

parent(tom,bob).

Объекты «Том» и «Боб» являются здесь константами, отношение с именем «parent» – это предикат. *Резольвента* – это множество целей, которые необходимо выполнить, чтобы получить ответ на поставленный вопрос.

Большинство ПРОЛОГ-трансляторов работают только с латинским алфавитом в качестве имен предикатов, поэтому для записи имен предикатов используется латинская нотация.

Совокупность фактов называется *базой данных*. Этот термин будет использоваться для объединения фактов при описании предметной области для решения конкретной задачи. Продолжив перечисление родственных отношений, получим некоторую базу следующего вида:

```
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,john).
```

Пусть имеется вопрос: «является ли Лиз родителем Боба? »

В терминах выбранной нотации ПРОЛОГа этот вопрос выглядит так:

?- parent(liz, bob).

Ответ «нет». Получила ПРОЛОГ-система его следующим образом:

1) из вопроса формируется резольвента, в нашем случае – parent(liz, bob);

2) выбирается первая цель из резольвенты (она у нас единственная);

3) просматривается база данных в попытке согласовать выбранную цель и факт из базы, в нашем случае такая попытка оказывается безуспешной.

Таким образом, для того чтобы получить положительный ответ на поставленный вопрос, база данных должна содержать факт, удовлетворяющий следующим условиям:

- имя факта и вопроса должны совпадать;
- должно совпадать число аргументов в вопросе и факте;
- вопрос и факт должны иметь одни и те же константы на тех же местах.

При выполнении этих условий вопрос удовлетворяет факту и, следовательно, ответ положителен. В нашем случае не нашлось факта, удовлетворяющего вопросу.

Теперь, положим, необходимо знать, родителем кого является Пэт. Перебор всех вопросов с указанием всех имен родственников, до получения ответа «да» – это не лучший вариант решения этой проблемы. Эту задачу можно решить, сформулировав вопрос следующим образом:

?- parent(pat, X).

Литера ***X*** означает в вопросе неизвестный заранее объект. Такого рода объекты называются переменными. *Переменная* в ПРОЛОГе – это последовательность букв и цифр, начинающаяся с прописной буквы или символа подчеркивания и содержащая только символы букв, цифр и подчеркивания. Переменная, состоящая из одного символа подчеркивания, – это *анонимная переменная*, использующаяся в предложении только один раз. В процессе поиска ответа на вопрос ПРОЛОГ-система просматривает базу данных и пытается найти эквивалентную замену переменной ***X*** объектом из базы, используя при этом алгоритм унификации. Ответом на вопрос: ?- parent(pat, X) будет X =john.

Переформулируем предыдущую задачу следующим образом: определить, кто родитель Джона. На вопрос

?- parent(X, john).

система ответит ***X=pat***. Таким образом, отношение parent ПРОЛОГ-система может использовать различными способами: если известны оба его аргумента, то проверяется выполнимость данного отношения на этих аргументах; если неизвестен один из аргументов, неважно какой из них, то система пытается найти эквивалентную замену переменной объектом из базы данных.

Во многих версиях ПРОЛОГа приняты следующие соглашения: заглавные буквы обозначают переменные, а строчные буквы, целые или действительные числа, или любые символы, заключенные в кавычки, обозначают постоянные значения – *константы*.

Предположим, что необходимо ответить на вопрос: «кто является родителем родителя Джона?» («кто дед Джона?»). В ПРОЛОГе это можно сделать в одном вопросе, воспользовавшись *общей переменной* и соединив оба вопроса, используя операцию конъюнкции: ?- **parent** (Y, john), **parent**(X, Y). Этот вопрос читается следующим образом: «Существуют ли такие ***x*** и ***y***, что **parent**(Y, john) и **parent**(X, Y) выполнимы одновременно?»

Рассмотрим действия ПРОЛОГ-системы при выполнении данного вопроса:

1) формируется резольвента из двух целей: **parent** (Y, john), **parent**(X, Y);

2) выбирается первая цель из резольвенты – parent (Y, john);

3) просматривается база данных в попытке согласовать выбранную цель и факт из базы. Попытка оказывается успешной, в базе найден факт parent(pat,john);

4) переменная Y с этого момента конкретизирована и ей соответствует значение pat;

5) первая цель резольвенты выполнена, система переходит к следующей цели – parent (X, pat);

6) просматривается база данных в попытке согласовать выбранную цель и факт из базы, попытка оказывается успешной, в базе найден факт parent(bob,pat);

7) переменной X присваивается значение bob;

8) в резольвенте отсутствуют невыполненные цели, значит, вопрос успешно выполнен, система выдает ответ: X = bob, Y = pat.

Сформулируем предыдущую задачу в общем виде:

«Определить кто чей предок».

Запрос к базе данных оформим в виде следующего правила:

predok(X, Y):- parent (X, Z), parent (Z, Y).

Представленное правило на естественном языке читается следующим образом: «если X – родитель Z, и Z – родитель Y, то X – **предок** (дед) Y ». Правило определяет предикат predok**,** имеющий два аргумента. Предикат **predok**(X, Y) истинен, если установлено взаимно-однозначное соответствие между именем ребенка Z (parent (X, **Z**)) и именем родителя Z (parent (**Z**, Y)).

Предыдущий вопрос будет записан следующим образом:

?- predok(X, john).

Действия ПРОЛОГ-системы здесь следующие:

1) формируется резольвента predok(X, john);

2) выбирается цель из резольвенты, т.е. predok(X, john);

3) делается попытка согласовать выбранную цель и заголовок правила из базы, попытка оказывается успешной;

4) переменная ***Y*** с этого момента конкретизирована, и ей соответствует значение john, переменная X не конкретизирована;

5) формируется новая резольвента parent(X, Z), parent(Z, john);

6) дальнейшие действия совпадают с действиями ПРОЛОГ-системы при решении предыдущей задачи с той лишь разницей, что вместо переменной X здесь используется Z.

Задание №1

Создайте предикат, проверяющий, являются ли два человека
- сестрами;
- братьями;
- дедушкой и внуком (внучкой);
- дядей и племянником (племянницей);
- супругами;
- родственниками.

Задание №2

Составьте базу из отношений «x читает курс y» (не менее 5) и «x работает на кафедре z». Вместо переменных x, y, z в базе укажите фамилии преподавателей, названия курсов и кафедр. Напишите следующие запросы:

1) Читает ли x курс y? (Например: читает ли istomina курс iis)
2) Какой курс читает x?
3) Кто читает курс y?
4) На какой кафедре работает x?
5) Кто на какой кафедре работает?
6) На какой кафедре читается курс y?

Лабораторная работа 3

Сложные термы.

Цель работы: Получение практических навыков работы со сложными термами.

Составим базу данных из фактов о месте и времени прочтения лекции, а также имени группы, в которой эта лекция читается. Время прочтения

лекции должно состоять из трех компонент: времени начала лекции, времени окончания лекции и дня недели:

time(«время начала лекции», «время окончания лекции», «день недели»),

здесь «время начала лекции» – это целое число;

«время окончания лекции» – также целое число;

«день недели» – слово.

Например, **time(**9, 11, vtornik).

Тогда база данных, будет состоять из фактов, подобных следующему

room_time_group(iis, 311, time(9, 11, vtornik), d_056).

Структурированные таким образом объекты называются *сложными термами*. Имя сложного терма называется *функтором*, компоненты – *аргументами*, число аргументов в терме – *арностью*. Сложный терм в программе ведет себя как единый объект данных. Так, например, на вопрос

?- **room_time_group**(iis, _, X, _).

система выдаст ответ: X = **time**(9, 11, vtornik).

Аргументы в сложном терме могут быть не только константами, но и переменными, например, вопрос «По каким дням недели читается курс “Искусственный интеллект”» будет выглядеть следующим образом:

?- **room_time_group**(iis, _ , **time**(_, _, X), _).

Ответ: X = vtornik.

Задание

Используя расписание движения поездов, создать базу данных, содержащую сложные термы.

Номер	Маршрут	отправления	стоянки	в пути	прибытия
010И	МОСКВА ЯР ИРКУТ ПАСС	02.18	00.23	23.33	01.51
904А	МОСКВА ЯР ВЛАДИВОСТ	03.33	02.07	35.50	15.23
068Ы	МОСКВА ЯР АБАКАН	04.49	00.23	25.33	06.22
044Э	МОСКВА ЯР ХАБАРОВС 1	09.59	00.24	25.26	11.25
076Э	МОСКВА КАЗ ТЫНДА	17.32	00.23	25.42	19.14
056Ы	МОСКВА ЯР КРАСНОЯР П	19.21	00.23	24.04	19.25
340Ч	МОСКВА ЯР ЧИТА 2	21.15	00.23	28.53	02.08
002М	МОСКВА ЯР ВЛАДИВОСТ	23.43	00.23	23.09	22.52

002Щ МОСКВА ЯР ВЛАДИВОСТ 23.43 00.23 23.09 22.52

УКАЗАНИЕ: при составлении БД использовать строчные английские буквы; при записи времени использовать формат xx_xx: 02_18.
Сформулируйте следующие запросы:

1) Получить маршрут и номер поезда, стоянка которого «00.23».
2) Получить время прибытия поезда «МОСКВА ЯР АБАКАН».
3) Получить время в пути всех поездов.

Лабораторная работа 4

Работа со списками.

Цель работы: Получение практических навыков работы списками.

В ПРОЛОГе имеется единственная структура данных – логический терм. С этой точки зрения, список – это терм арностью два, в котором второй аргумент является также списком.

С математической точки зрения, список – это конечная упорядоченная последовательность элементов. Список в языках программирования – это бинарная структура или специальный вид сложного терма, состоящий из головы списка и хвоста списка, где голова – это элемент списка, а хвост – это список. Элементы списка, в свою очередь, сами могут быть списками. Список принято обозначать следующим образом:

[X | L],

где ***X*** – голова, ***L*** – хвост, операция «|» обозначает слияние элемента-головы и элемента-хвоста списка в единый список. Операцию «|» можно интерпретировать и как операцию разбиения списка на голову и хвост. Таким образом, список в ПРОЛОГе – это такая структура, в которой операции включения /исключения производятся только в начале этой структуры. Для элементов списка справедливо следующее правило: «Первым пришел – последним ушел».

На вопрос

?- L = [a | [b, c, d]]. (1)

будет дан ответ **L = [a, b, c, d].**

При построении списков используется константный символ **[]**, обозначающий пустой список. Важное свойство пустого списка – его нельзя разделить на голову и хвост, попытка выполнить такую операцию приведет к неудаче.

Дадим **рекурсивное определение списка**:

• **[]** – это список;

• пусть ***L*** – список, ***A*** – терм, тогда **[*A* | *L*]** – список.

На вопрос

?-L = [a , b, c, d | []]. (2)

будет дан ответ **L = [a, b, c, d].**

Элементами списка могут быть не только константы, но любые термы – переменные, структуры, включая другие списки. Важным свойством списка является его динамичность, т.е. отсутствие необходимости заранее описывать размер списка. Переменные в списке ничем не отличаются от переменных в любой другой структуре, с ними можно проводить те же действия, что и с переменными вне списка.

На вопрос

?- L = [a , b, c, d], L = [X | Y], Y = [V, Z | W]. (3)

будет дан ответ

L = [a, b, c, d], X = a, Y = [b, c, d], V = b, Z = c, W = [d].

На вопрос

?- L = [], L = [X | Y]. (4)

будет дан ответ «нет», так как пустой список нельзя разделить на голову и хвост.

Таким образом, список в ПРОЛОГ-программе, если он не пуст, можно представить различными способами:

- перечислением всех элементов списка – **[a, b, c, d], [X, Y, Z, V, W];**
- перечислением нужного количества элементов начала списка и ссылкой на его хвост – **[a , b, c | L]** или **[X , Y | L]**;

Фундаментальной операцией на списках является определение вхождения отдельного элемента в список, дадим имя этому отношению **member(X, L),** где ***X*** – это элемент, ***L*** – это список. Сформулируем ***граничное условие*:** элемент содержится в списке, если он совпадает с головой этого списка. ***Рекурсивное условие*:** элемент содержится в списке, если он является элементом хвоста этого списка.

Программа, рекурсивно определяющая отношение принадлежности элемента списку, имеет вид:

member(X, [X | _]). (5)

member(X, [_ | Y]):- member(X, Y). (6)

Строка программы (5) представляет собой сформулированное выше граничное условие. Строка программы (6) представляет собой сформулированное выше рекурсивное условие.

Ответом на вопрос

?- member(a, [b, a, c]). (7)

будет «да».

Рассмотрим процесс получения данного ответа. Начальная резольвента (или цель) ***member(a, [b, a, c]).*** В программе выбирается первое предложение и делается попытка унифицировать его с резольвентой, т.е найти в голове списка символ ***a***. Попытка заканчивается неудачей, так как переменной ***X*** одновременно не может быть присвоено значение ***a*** (первый аргумент в предикате member) и ***b*** (голова списка второго аргумента предиката member). Далее в программе выбирается второе предложение и делается попытка унифицировать его с резольвентой. Попытка удачна, новая и единственная резольвента – ***member(a, [a,c]).*** В программе выбирается первое предложение, и оно успешно унифицируется с резольвентой. Первое предложение программы не имеет тела, резольвента пуста, решение найдено. Но резольвента ***member(a, [a,c])*** может быть унифицирована и со вторым предложением программы, после унификации получаем новую резольвенту ***member(a, [c]).*** Данная резольвента не может быть унифицирована с первым предложением программы, а вот со вторым предложением унификация заканчивается успешно, новая резольвента – ***member(a,*** **[]** ***),*** которая не может быть унифицирована ни с первым, ни со вторым предложением. Таким образом, был получен один положительный ответ. Графическое представление поиска решения приведено на рис. 1.

Операция соединения двух списков для получения третьего: ***append(L1, L2, L3),*** здесь ***L3*** – это конкатенация списков ***L1*** и ***L2***. ***Граничное условие***: результатом соединения пустого списка со списком ***L*** будет список ***L***. ***Рекурсивное условие*****:** результатом соединения списка ***L1*** и списка ***L2*** является список ***L3***; соединение выполняется следующим образом: голова списка ***L1*** добавляется к хвосту списка ***L3***.

Ниже приведена программа, задающая заданное отношение:

append([], L, L). //граничное условие

append([X | XL], YL, [X | ZL]):- append(XL, YL, ZL). //рекурсивное условие

Ответом на вопрос:

?- **append([a, b, c], [1, 2], Z).**

будет **Z = [a, b, c, 1, 2].**

Отношение ***append*** может быть использовано и для выполнения операции обратной соединению двух списков, а именно для разбиения списка на возможные составляющие. Ответом на вопрос

?- **append(A, B, [a, b, c]).**

будет:

B = [a, b, c],

A =[];
B = [b, c],
A =[a];
B = [c]
A =[a,b];
B = []
A =[a,b,c];
false.

Рассмотрим пример программы проверки существования в двух списках хотя бы одного общего элемента:

intersect(XL, YL):- member(X, XL), member(X, YL).

Первая цель ***member*** в теле этого предложения генерирует элементы из первого списка, а с помощью второй цели ***member*** проверяется, входят ли эти элементы во второй список.

Рассмотрим выполнение предиката, определяющего принадлежность элемента списку:

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

Вопрос: ?- member(a, [a, b, a, c, a]). сгенерирует три возможных решения.

Очевидно, если предикат используется только для определения вхождения элемента в список, то нет нужды во всех трех решениях, достаточно ответа да/нет. В ПРОЛОГе существует механизм, позволяющий отбросить ненужные решения. Механизм этот называется «отсечение». Отсечение – это встроенный предикат, затрагивающий процедурное поведение программ. Синтаксически в правиле отсечение задается как предикат «!» (CUT). Этот предикат всегда выполним, однако он влияет на процесс последующего возврата.

Можно предотвратить ненужный перебор в программе, определяющей принадлежность элемента списку, если изменить первое предложение, добавив в него отсечение следующим образом:

member(X, [X | _]):- !.
member(X, [_ | Y]):- member(X, Y).

Заданное таким образом отношение сгенерирует единственное решение.

Эффективность программы, в которую включено отсечение, может возрасти по двум причинам:

- возрастет скорость выполнения, в силу того что будут отсутствовать попытки унификации целей, о которых заранее известно, что они приведут к неуспеху;

- уменьшится объем занимаемой памяти, так как не будут запоминаться точки возврата для последующего анализа.

Кроме того, применяя отсечение, можно повысить выразительность языка при описании взаимоисключающих правил. Однако применение отсечения приводит и к определенным проблемам, а именно, программист должен точно знать, как будут использоваться правила, содержащие отсечения.

Отсечение используется в следующих трех случаях:

1) когда необходимо указать ПРОЛОГ-системе, что найдено нужное правило для выполнения;

2) когда необходимо указать ПРОЛОГ-системе на немедленное прекращение поиска решений без попытки найти альтернативные решения;

3) когда необходимо указать ПРОЛОГ-системе закончить порождение альтернативных решений.

Библиотека предикатов

При работе со списками также используются следующие предикаты:

select(L1, elem, L2)

предикат выполняет следующие действия:

1) из исходного списка ***L1*** удаляется элемент ***elem***, результат записывается в список ***L2;***

2) в начало списка ***L2*** добавляется элемент ***elem***, результат записывается в список ***L1***.

delete(L1, elem, L2)

предикат выполняет удаление всех elem из списка L1. L2 – результирующий список.

reverse(L, R)

предикат выполняет обращение списка L в список R.

last(L, elem)

предикат определяет последний элемент списка L.

nextto(Y,X,L)

предикат определяет элемент Y непосредственно следующий за X в списке L.

sumlist(L,Sum)

определяет сумму всех элементов, входящих в список L.

min_list(L,Min)

определяет min число списка L.

max_list(L,Max)

определяет max число списка L.

permutation(L1,L2)

позволяет получить все возможные перестановки списка.

Задание 1

1 Даны два списка: [a,b] и [c,d]. Получите список [a,b,c,d].

2. Дан список [1,2,3]. Получите все возможные два подсписка.

3. Дан список [a1,a2,a3]. Удалите элемент a2 из списка. Добавьте элемент b в список.

4. Дан список [d, e, a, b, c, d], удалите элемент **d** из списка.

5. Получите обращение списка [1,2,3,4].

6. Найдите последний элемент списка [d,f,g,h,k,l].

7. Получите все возможные перестановки списка [d,f,g,h].

8. Напишите предикат p(L, elem), позволяющий определить elem - предпоследний элемент списка L, имеющего не менее двух элементов.

9. Напишите предикат p(L1,L2,N), позволяющий формировать подсписок L2 из первых N элементов заданного списка L1.

10. Напишите предикат p(X, N, L), который создает список L, состоящий из N элементов X.

11. Напишите предикат p(N,K,L,L1,L2), предназначенный для формирования списка из последних элементов исходного списка (N - длина списка L; K – общее количество элементов, подлежащих перезаписи в результирующий список L2; L1 – промежуточный список).

12. Напишите предикат count(L,N), позволяющий определить число элементов N в списке L.

13. Напишите предикат p(L,N,El), который позволяет определить элемент El, имеющий номер N в списке L.

Задание 2

1. Создайте предикат, осуществляющий поэлементное перемножение соответствующих элементов двух исходных списков.

2. Создайте предикат, осуществляющий подсчет числа вхождений каждого элемента исходного списка. Ответом должен быть список пар, в которых первая компонента - элемент исходного списка, вторая - число его вхождений в первоначальный список.

3. Создайте предикат, определяющий первую позицию подсписка в списке.

Создайте предикат, добавляющий элементы одного списка во второй список, начиная с заданной позиции.

4. Создайте предикат, возвращающий по списку и двум числам M и N подсписок исходного списка, состоящий из элементов с номерами от M до N.

5. Создайте предикат, формирующий список простых чисел, не превосходящих данного числа.

Создайте предикат, транспонирующий матрицу, заданную списком списков.

Лабораторная работа 5
Классифицирующие системы.

Цель работы: Получение практических навыков работы со сложными термами и списками. Разработка пролог-программы, позволяющей классифицировать объекты.

В общем виде задачу классификации можно сформулировать следующим образом:

1) имеется конечное множество объектов, которые можно описать определенным набором признаков,

2) каждый объект принадлежит одному классу из некоторого фиксированного множества,

3) в задаче классификации по заданному набору признаков необходимо определить, к какому классу принадлежит объект.

Задание 1

1. Создайте "определитель пород собак".

2. Создайте программу, позволяющую диагностировать заболевание по симптомам.

3. Создайте программу, позволяющую диагностировать неисправность компьютера.

4. Создайте программу, помогающую школьнику определиться с выбором будущей профессии.

Задание 2

5. Создайте программу, осуществляющую поиск пути в лабиринте. Задача заключается в следующем. Имеется описание лабиринта в виде набора координат стен, а также координаты текущей позиции в лабиринте и координаты выхода. Требуется найти путь от текущей позиции до выхода. В качестве дополнительного задания можно подсчитать длину пройденного пути.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем принципиально отличаются языки логического программирования от традиционных языков программирования?
2. Какую роль играет переменная в логическом языке программирования? Какова область ее действия?
3. Что такое цель в логическом программировании?
4. Как задается и выполняется ПРОЛОГ-программа? Перечислите типы ответов на вопросы.
5. Чем определяется важность рекурсии в логическом программировании?
6. В чем заключается принципиальное отличие итерации от рекурсии?
7. Когда и почему в логическом программировании применяется отсечение?

ЛИТЕРАТУРА

1. Зюзьков В. М. Искусственный интеллект : учебное пособие - Томск : НТЛ, 2007. - 152 с.
2. Ходашинский И. А.Язык ПРОЛОГ в примерах и задачах : Учебное пособие для вузов /. Томск : ТУСУР, 2006. - 279[1] с. : ил. - **ISBN** 5-86889-291-7.
3. Ходашинский И. А. Методы искусственного интеллекта, базы знаний, экспертные системы : Учебное пособие / - Томск : ТУСУР, 2002. - 140 с.