|  |
| --- |
| **Отчет по лабораторной работе № 3****«Логическое программирование на языке GNU Prolog»** |
| дата | Оценка(max 5) | Бонус за сложность | подпись |

Цели работы:

Получить практические навыки применения систем и языков логического программирования для построения систем, основанных на знаниях. Создать экспертную систему по диагностике и ремонту компьютеров с использованием среды GNU Prolog

Задачи работы:

На языке Пролог реализовать базу знаний на выбранную тему, с использованием прямого либо обратного логического вывода. База знаний должна обязательно включать несколько уровней рассуждений (т.е. окончательные выводы не должны напрямую следовать из комбинаций входных данных, необходимо использовать промежуточные выводы) и демонстрировать некоторую интеллектуальность в принятии решений. Ориентировочное количество правил если-то – около 50 шт.

**Задание повышенной сложности (бонус за сложность – 10 баллов):**

Разработать полнофункциональную экспертную систему на языке Пролог (прототип модули ЭС на Си, файл lab1.doc) для синтеза технических решений по варианту комплексного заданя.

**Краткий конспект теоретической части** (изучите теоретическую часть файл lab3.doc)

|  |
| --- |
| Что такое язык ПРОЛОГ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Типы данных языка Пролога и их свойства \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Синтаксис программ на языке Пролог \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Понятие списков и их использование \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**PROLOG** (PROgramming in LOGic) – это язык программирования, используемый для решения задач, в которых действуют объекты и отношения между этими объектами. Предложен в районе 1972 года Alain Colmerauer и Philippe Roussel и на данный момент не стандартизирован. Существует много реализаций данного языка: Visual Prolog, PDC Prolog, Turbo Prolog, Prova, InterProlog, GNU Prolog. В рамках занятий используется GNU Prolog (доступен для скачивания на <http://www.gprolog.org/>). Возможно выполнение заданий на любой другой реализации языка.

На рис. 1. Представлен внешний вид главного окна GNU Prolog.



Рис. 1 – Внешний вид главного окна GNU Prolog.

**Основные команды GNUProlog:**

* **[**file\_name**].** - загрузить файл с кодом программы (файл указывается без расширения «.pl»)‏
* **trace.** - включить режим трассировки программы.
* **notrace.** - выключить режим трассировки программы.
* **halt.** - завершить выполнение программы.
* **[user].** - войти в режим интерактивного ввода программы. Control+D выход из режима интерактивного ввода программы

**Задание:** изучите теоретическую часть по ЛР, представленной на сервере http://oracle.iu4.bmstu.ru, скачайте и установите GNUProlog по следующей ссылке: <http://www.gprolog.org/setup-gprolog-1.3.1.exe>.

**Синтаксис языка Prolog**

**Термы** — единица данных, объект данных в Prolog. Подразделяются на:

* Переменные
* Константы
* Числа
* Атомы
* Строки
* Списки
* Структуры
* Составные термы

**Переменными** в Прологе являются строки символов, цифр и символа подчеркивания, начинающиеся с заглавной буквы или символа подчеркивания. В случае, если имя переменной состоит из одного символа “\_”, переменная называется *анонимной*.

Область действия всех переменных, кроме анонимной, заключена в пределах высказывания. Анонимная переменная применяется, когда ее значение не используется в программе. Неоднократное употребление безымянной переменной в одном выражении применяется для того, чтобы подчеркнуть наличие переменных при отсутствии их специфической значимости.

Таким образом, в прологе переменные могут быть конкретизированы только один раз, например, выражение вида N = N + 1 не имеет смысла. Подобные вещи в основном реализуются за счёт рекурсии.

**Числа** – в привычном понимании целые и действительные числа.

**Пример 1.** Вклейте результат выполнения произвольного запроса с объявлением чисел.

|  |  |
| --- | --- |
| **?- *A* = 12, *B* = 0.5e4.***A = 12B = 5000.0* |  |

Пролог не производит вычисления автоматически, т.е. «2+3» будет восприниматься как терм. Для осуществления вычислений используется оператор «is».

**Пример 2.** Вклейте результаты выполнения нижеприведённых запросов

|  |  |
| --- | --- |
| **?- X = 2+3.****?- X is 2+3.** |  |
| **Основные арифметические операторы:****+** сложение**-** вычитание**\*** умножение**/** деление**//** целочисленное деление**mod** остаток от деления**\*\*** возведение в степень | **Операторы сравнения:****=:=** равенство;**=\=** неравенство;**<** меньше;**>** больше;**>=** больше либо равно;**=<** меньше либо равно. |

**Задание 1.** Разработайте запрос для вычисления среднего арифметического двух чисел.

|  |
| --- |
|  |

**Задание 2.** Разработайте запрос для вычисления дискриминанта квадратного уравнения.

|  |
| --- |
|  |

**Атомы** – это любая последовательность символов, заключённая в одинарные кавычки (апострофы).

**Пример 3.** Вклейте результат выполнения приведённого запроса.

|  |
| --- |
| **?- A = abc, B = 'Hello World'.***A = abcB = 'Hello World‘* |
|  |

**Строки –** списки кодов символов в кодировке ASCII, из которых состоит данное выражение. Таким образом, к строкам применимы все те же операции, что и к спискам.

**Пример 4.** Вклейте результат выполнения приведённого запроса.

|  |
| --- |
| **?- S = ”Hello World!”.***S = [72,101,108,108,111,32,87,111,114,108,100,33]* |
|  |

**Список**- последовательность термов, заключенная в квадратные скобки.

**Пример 5.** Вклейте результат выполнения приведённого запроса.

|  |
| --- |
| **?- A=[], B=[a, tro, 123, lolo, ”Hello!”], C=[A,B].***A = []**B = [1,tro,123,lolo,[72,101,108,108,111,33]]**C = [[],[1,tro,123,lolo,[72,101,108,108,111,33]]]* |
|  |

Это может быть любой пустой список, не содержащий ни одного элемента, либо структура, имеющая два компонента - **голову** и **хвост**. Пустой список не имеет ни головы, ни хвоста. Для расщепления списка в Прологе введена специальная форма - символ “|”:

|  |  |  |
| --- | --- | --- |
| **Выражение** | **Голова** | **Хвост** |
| [a, b, c] | a | [b, c] |
| [a] | A | [ ] |
| [ ] | Нет головы | Нет хвоста |
| [[a, b], c] | [a, b] | [c] |
| [a, [b,c]] | A | [[b, c]] |

**Пример 6.** Вклейте результат выполнения приведённого запроса.

|  |
| --- |
| **?- Head = iu4, Tail = [f,o,r,e,v,e,r], Fool = [Head | Tail].***Fool = [iu4,f,o,r,e,v,e,r]**Head = iu4**Tail = [f,o,r,e,v,e,r]* |
|  |

**Задание 3.** Разработайте запрос, выделяющий голову и хвост из списка *Fool* из примера 6.

|  |
| --- |
|  |

**Задание 4.** Разработайте запроc, присваивающий переменной T последние 3 символа списка *Fool* из примера 6.

|  |
| --- |
|  |

**Знак "="** обозначает не императивное равенство (присвоение), а ***унификацию*** (что в других языках называется сопоставление с образцом), т.е. сопоставление левой и правой части и в случае удачного сопоставления конкретизация неизвестных значений.

**Пример 7.** Вклейте результат выполнения приведённого запроса.

|  |
| --- |
| **?- tro=lolo.***no %атомы несовместимы***?- trololo=trololo.***yes %сопоставление удачно***?- 1=X.***X=1**yes %атомы несовместимы***?-[1,2| T] = [1,B,3,4].***T = [3,4]B = 2* |
|  |

**Логические операторы.**

Функтор “**,**” – логическое **«И»**.

**Пример 8.** Вклейте результат выполнения приведённого запроса.

|  |  |
| --- | --- |
| **?- A = 2, B = 3, A = B.***no* |  |

Функтор “**;**” – логическое **«ИЛИ»**.

**Пример 9.** Вклейте результат выполнения приведённого запроса.

|  |  |
| --- | --- |
| **?- A = 2; A = 5.***A = 2**A = 5* |  |

В данном случае система выдаёт неопределённый ответ. A может принимать как значение 2, так и 5.

Одной из особенностей языка Пролог является **backtracking** (перебор с возвратом). Добавим конкретики в предыдущий пример.

**Пример 10.** Вклейте результат выполнения приведённого запроса.

|  |  |
| --- | --- |
| **?- (A=2; A=5), A mod 2 =:= 0.***A = 2**no* |  |

Для данного примера система рассуждает следующим образом — допустим A=2, проведем проверку на A mod 2 =:= 0 — это выполняется => сопоставление удачно — выполняется конкретизация (A=2), далее происходит откат (возврат) к первой части утверждения и проверяется гипотеза A = 5, что является ложью и система возвращает «no». Возврат после первого удачного сопоставления можно отменить с помощью оператора отсечения "!".

**Задание 5.** Модифицируйте пример 10 таким образом, чтобы система выходила из цикла сразу же после первого удачного сопоставления.

|  |
| --- |
|  |

**Программа на языке Prolog**

До этого мы имели дело с интерактивным режимом. Программа должна храниться в текстовом файле с расширением “.pl”. Для загрузки файла используется следующая команда:

**[***file\_name***].**

где *file\_name* – имя файла программы **без расширения “.pl”**.

Программа на Прологе обычно представляет собой совокупность фактов и предикатов. Таким образом, рассмотренные далее структуры должны быть занесены в текстовый файл, после чего загружены для исполнения в оболочке.

**Структуры**

**Структура** – конструкция следующего вида:

food(fruit, banana). %банан – это фрукт, он съедобен

food(fruit, kiwi). %киви – это фрукт, он съедобен

food(vegetable, cabbage). %капуста – это овощ, она съедобна

Структура в прологе представляется **функтором** (имя структуры - food) и **параметрами** (fruit, banana). Число параметров называется **арностью**функтора.

**Утверждения** подразделяются на факты и предикаты (правила).

**Факт** - это одиночная цель, которая, безусловно, истинна:

primacy(man). % человек - примат

primacy(monkey). % обезьяна – примат

**Предикат (правило)** состоит из одной головной цели и одной или более хвостовых целей, которые истинны при некоторых условиях:

can\_eat(X) :- primacy (X).

Т.е., чтобы что-то могло есть, оно должно быть приматом.

Примеры описания фактов и предикатов представлены см. на следующей странице.

**Задание 6.** Разработайте базу знаний к Вашему примеру из Лабораторной работы №1

|  |
| --- |
| *% типы живых существ: человек, обезьяна*primacy(man).primacy(monkey).*% имена существ и принадлежность их к виду*exemplar(monkey,chicky).exemplar(man,mark).exemplar(man,yud).*% кто и где живёт*living(yud,russia).living(mark,america).living(chicky,africa).*% кто что ест*eating(chicky,banana).eating(chicky,kiwi).eating(yud,apple).eating(yud,lemon).eating(mark,onion).eating(mark,potato).eating(mark,cabbage).*% предикат, показывающий инфу о живых существах**% X - что за примат**% Y - его имя**% Z - где живёт**% E - что ест*alive(X,Y,Z,E):- primacy(X), exemplar(X,Y), living(Y,Z), eating(Y,E).*% еда: тип (фрукт/овощ), имя*food(fruit,banana).food(fruit,lemon).food(fruit,apple).food(fruit, kiwi).food(vegetable,onion).food(vegetable,potato).food(vegetable,cabbage).*% подробная инфа о еде: имя, вкус, цвет, форма.*description(banana,sweet,yellow,long).description(lemon,sour,yellow,round).description(apple,sweet,red,round).description(kiwi,sour,green,round).description(onion,bitter,yellow,round).description(potato,sweet,brown,round).description(cabbage,sweet,green,round).*% предикат, показывающий инфу о хавчике**% X - тип еды (фрукт/овощ)**% Y - название еды**% A - вкус**% B - цвет**% C - форма*havchik(X,Y,A,B,C):- food(X,Y), description(Y,A,B,C). |

Вставьте скриншоты результатов выполнения запросов к вашей базе знаний.

|  |
| --- |
|  |

|  |
| --- |
|  |

**Трассировка программы**

**Трассировка –** пошаговое выполнение программы. Для входа в режим трассировки необходимо выполнить одно из следующих действий:

* нажать “**Ctrl+C**”, затем ввести команду “**t**”;
* выполнить команду “**trace.**”

В качестве примера, рассмотрим трассировку программы быстрой сортировки.

 qsort( IN, OUT ) :- qsort\_dl( IN, OUT, [] ).

 qsort\_dl( [X|M], R, R0 ) :-

 partition( M, X, M1, M2 ),

 qsort\_dl( M2, R1, R0 ),

 qsort\_dl( M1, R, [X|R1] ).

 qsort\_dl( [], R, R ).

 partition( [X|M], Y, [X|M1], M2 ) :-

 X<Y, !, partition( M, Y, M1, M2 ).

 partition( [X|M], Y, M1, [X|M2] ) :-

 partition( M, Y, M1, M2 ).

 partition( [], \_, [], [] ).

**Загрузка программы**



**Режим пошагового выполнения**



**Задание 7.** Выполните трассировку программы к своему примеру и вставьте скриншот с результатом её выполнения.

|  |
| --- |
|  |

**Контрольные вопросы**

1. Формальная логика. Понятие. Суждение. Высказывание. Формальный язык, грамматика формального языка.

2. Логика предикатов. Интерпретация и унификация. Фразы Хорна. Принцип резолюции.

3. Парадигма декларативного программирования. Представление знаний с помощью фактов и правил. Управление вычислениями.

4. Простые и составные запросы. Понятие анонимной переменной.

5. Объекты данных. Структурирование множества объектов данных.

6. Рекурсивные вычисления.

7. Итерационные вычисления.

8. Рекурсивные структуры данных: списки. Способы обработки и примеры использования.

9. Рекурсивные структуры данных: деревья. Способы обработки и примеры использования.

10. Встроенные предикаты обработки символьных данных.

11. Встроенные предикаты управления вычислениями.

12. Способы представления баз знаний.

13. Представление баз знаний с использованием рекурсивных структур данных.

14. Создание графических изображений средствами языка Пролог.

15. Поиск на графах пространства состояний. Поиск в глубину и ширину.

16. Поиск на графах пространства состояний. Эвристический поиск.

17. Применение Пролога в естествознании.

18. Экспертные системы на правилах.

19. Экспертные системы на логике.

**Список литературы**

1. Адаменко А. Логическое программирование и Visual Prolog (+CD-ROM). BHV.

2. Попов Э.В. Экспертные системы: решение информационных задач в диалоге с ЭВМ. М.: Наука, 1987. 283 с.

3. Астаппн Н.И., Матвеев М.Г. Синтез задач ситуационного управления перерабатывающим предприятием // Научно-техн. сб. мясной и холодильной промышленности РА сельскохоз. наук. 1994. № 2. С. 19-22.

4. Айзерман М.А., Алесекров Ф.Т. Выбор вариантов. Основы теории. М.: Наука, 1990. 240 с.

5. Месарович М., Такахара Я. Общая теория систем: математичес­кие основы. М.: Мир, 1978. 311с.

6. Шоломов Л.А. Логические методы исследования дискретных моделей выбора. М.: Наука, 1989. 288 с.

7. Матвеев М.Г., Сысоев В.В. Концепция информационных технологий управления перерабатывающими производствами // Информационная бионика и моделирование. М.: ГОСИФТП РАН, 1995. С. 25-31.

8. Минский М. Фреймы для представления знаний. М., 1979.

9. Робототехника и гибкие автоматизированные производства: В 9 кн. Кн. б. Техническая имитация интеллекта / В.М. Назаретов. Д.П. Ким. Под ред. И.М. Макарова. М.: Высш. шк., 1986. 144 с.

10. Марселлус Д. Программирование экспертных систем на Турбо-ПРОЛОГЕ. М.: Финансы и статистика, 1994. 256 с.

Иван Братко «Программирование на языке Пролог для систем искусственного интеллекта»

11. Марселлус «Программирование экспертных систем на Прологе»

12. Нейлор «Как построить свою экспертную систему»