

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

КАФЕДРА КОМПЬЮТЕРНЫХ СИСТЕМ И ПРОГРАММНЫХ ТЕХНОЛОГИЙ

**Отчёт по лабораторной работе №4**

**Курс: «Язык искусственного интеллекта PROLOG»**

Выполнил студент:

Ерниязов Т.Е.

Группа: 13541/2

Проверил:

Сазанов А.М.

Санкт-Петербург  
2018 г.

# Содержание

<b>1</b>	<b>Лабораторная работа №4</b>	<b>2</b>
1.1	Цель работы . . . . .	2
1.2	Программа работы . . . . .	2
1.3	Ход работы . . . . .	3
1.3.1	Запустите демонстрационный проект family1 . . . . .	3
1.3.2	Постройте генеалогическое дерево для данного примера на основе результатов выполнения программы и исходного кода программы . . . . .	3
1.3.3	Построить описание онтологии из данного примера на естественном языке . . . . .	3
1.3.4	Построить концептуальную карту (семантическую сеть), описывающую данный пример . . . . .	4
1.3.5	Создать проекты 1-21 для каждого из примеров в пособии привести листинги результатов работы каждой из программ в ответ на запросы пользователя . . . . .	4
1.3.6	Выполнить индивидуальное задание из пособия Буракова С.В. . . . .	26
1.3.7	Выполнить индивидуальное задание из пособия Седана С.Н. . . . .	27
1.4	Вывод . . . . .	29
1.5	Список литературы . . . . .	30

# Лабораторная работа №4

## 1.1 Цель работы

Ознакомиться с языком PROLOG и средой разработки Visual Prolog.

## 1.2 Программа работы

1. Получите начальное представление о синтаксисе и семантике базовых конструкций языка PROLOG, ознакомившись с разделами 1-5 методического пособия [1]
2. Создайте проект в оболочке Visual Prolog 7.3., как это показано в примере [2]
3. Удалить проект, созданный в предыдущем пункте и запустить демонстрационный проект family1 в оболочке Visual Prolog 7.3.
4. Постройте генеалогическое дерево для данного примера на основе результатов выполнения программы и исходного кода программы.
5. Построить описание онтологии из данного примера на естественном языке.
6. Построить концептуальную карту (семантическую сеть), описывающую данный пример.
7. Создать проекты 1-21 для каждого из примеров в пособии из п.1 и привести листинги результатов работы каждой из программ в ответ на запросы пользователя.
8. Выполнить индивидуальное задание для варианта 9 [3]
9. Изучить 1-2 лабы по методичке [1] Согласно своему варианту решить задачу с помощью PROLOG.
10. В выводах отразить следующее:
  - В чем Плюсы и минусы языка Prolog?
  - Какие еще языки используются для разработки ИИ, приведите примеры (НЕ МЕНЕЕ 2-х) проектов, языков и краткое описание проектов. (Альтернативы PROLOG)
  - Решаема ли проблема комбинаторного взрыва, пути решения?
  - Корректно ли по-вашему в принципе разработка языка ИИ? Что он должен из себя представлять?
  - Можно ли разработать ИИ не понимая, как он работает, должны ли мы понимать, как он работает, думает, рассуждает?

## 1.3 Ход работы

### 1.3.1 Запустите демонстрационный проект family1

```
father test
John is the father of Bill
Bill is the father of Pam

grandFather test
John is the grandfather of Pam

ancestor of Pam test
Bill is the ancestor of Pam
John is the ancestor of Pam
Judith is the ancestor of Pam
End of test
```

Рис. 1.1: Запуск демонстрационного проекта

### 1.3.2 Постройте генеалогическое дерево для данного примера на основе результатов выполнения программы и исходного кода программы

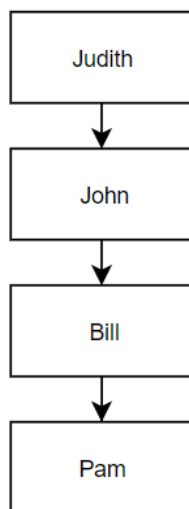


Рис. 1.2: Генеалогическое дерево для конкретного примера

### 1.3.3 Построить описание онтологии из данного примера на естественном языке

Программа определяет степень родства между людьми. Входными данными для программы являются:

- Имя и пол человека.
- Родительские связи.

На основании этих данных программа определяет:

- Если человек является родителем и мужчиной, то он — отец.
- Если человек является родителем другого родителя и мужчиной, то он — дедушка.
- Если человек связан цепочкой родительских связей с другим человеком, то они — родственники.

### 1.3.4 Построить концептуальную карту (семантическую сеть), описывающую данный пример

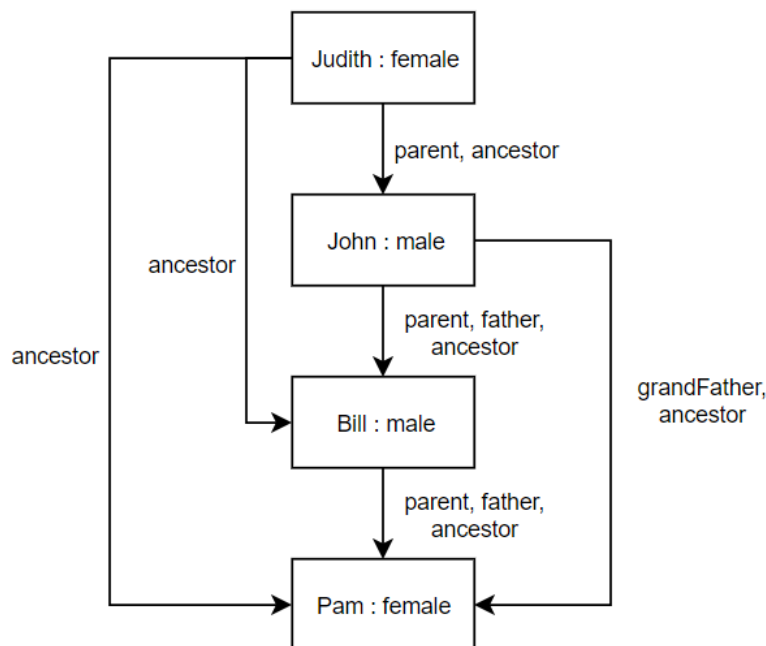


Рис. 1.3: Концептуальная карта, описывающая данный пример

### 1.3.5 Создать проекты 1-21 для каждого из примеров в пособии привести листинги результатов работы каждой из программ в ответ на запросы пользователя

#### Задание 1

В программе выводятся все данные предиката situ. Кроме того, определены два правила, которые позволяют отнести города из России и Польши к городам Европы.

```
1 implement main
2     open core
3
4 class predicates
5     situ : (string Gorod, string Strana) nondeterm anyflow.
6 clauses
7     situ("london", "england").
8     situ("petersburg", "russia").
9     situ("kiev", "ukraine").
10    situ("pekin", "asia").
11    situ("warszawa", "poland").
12    situ("berlin", "europe").
13    situ(X, "europe") :-
14        situ(X, "russia").
15    situ(X, "europe") :-
16        situ(X, "poland").
17
18 clauses
19     run() :-
20         console::init(),
21         situ(X, Y),
22         stdIO::writef("%s - %s\n", X, Y),
23         fail.
24
25     run().
26
```

```

27 end implement main
28
29 goal
30   console :: runUtf8(main :: run) .

```

```

london - england
petersburg - russia
kiev - ukraine
pekin - asia
warszawa - poland
berlin - europe
petersburg - europe
warszawa - europe

```

Рис. 1.4: Результаты выполнения задания

## Задание 2

Составные объекты позволяют описывать иерархические структуры, в которых описание одного предиката включает в себя описание других предикатов. Данная программа иллюстрирует использование составных объектов:

```

1  implement main
2    open core
3
4  domains
5    collector = symbol.
6    title = symbol.
7    author = symbol.
8    publisher = symbol.
9    year = integer.
10   personal_library = book(title , author , publication).
11   publication = publication(publisher , year).
12
13 class predicates
14   collection : (collector [out], personal_library [out]).
15
16 clauses
17   collection("Ivanov", book("War and Peace", "Lev Tolstoy", publication("
18   prosveshcheniye", 1990))).
19
20 clauses
21   run() :-
22     console :: init(),
23     collection(X, Y),
24     stdIO :: writef("% - %\n", X, Y),
25     fail.
26   run().
27
28 end implement main
29
30 goal
31   console :: runUtf8(main :: run) .

```

```

Ivanov - book("War and Peace","Lev Tolstoy",publication("prosvesh

```

Рис. 1.5: Результаты выполнения задания

### Задание 3

Программа описывает задачу о семейных отношениях. Имеются исходные данные об отцовстве:

1. Иван – отец Игоря.
2. Иван – отец Сидора.
3. Сидор – отец Лизы.

Требуется определить, есть ли братья у Игоря.

```
1 implement main
2   open core
3
4 domains
5   person = symbol.
6
7 class predicates
8   otec : (person, person) nondeterm anyflow.
9   man : (person) nondeterm.
10  brat : (person [out], person [out]) nondeterm.
11
12 clauses
13   man(X) :-
14     otec(X, _).
15
16   brat(X, Y) :-
17     otec(Z, Y),
18     otec(Z, X),
19     man(X),
20     X <> Y.
21
22   otec("ivan", "igor").
23   otec("ivan", "sidor").
24   otec("sidor", "lisa").
25
26 clauses
27   run() :-
28     console::init(),
29     brat(X, Y),
30     stdIO::writef("% is brother of %\n", X, Y),
31     fail.
32
33   run().
34
35 end implement main
36
37 goal
38   console::runUtf8(main::run).
```

A terminal window with a black background and white text. The text 'sidor is brother of igor' is displayed on a single line. A small cursor is visible at the end of the line.

Рис. 1.6: Результаты выполнения задания

### Задание 4

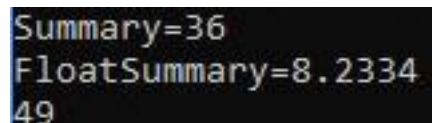
Помимо встроенных функций для арифметических выражений можно использовать собственные предикаты. В данной программе осуществляется поиск суммы целых чисел, суммы вещественных чисел и максимума из двух вещественных чисел.

```
1 implement main
2   open core
3
```

```

4 class predicates
5     add : (integer, integer).
6     fadd : (real, real).
7     maximum : (real, real, real [out]) nondeterm.
8
9 clauses
10     add(X, Y) :-
11         Z = X + Y,
12         stdIO :: write("Summary=", Z),
13         stdIO :: nl.
14
15     fadd(X, Y) :-
16         Z = X + Y,
17         stdIO :: write("FloatSummary=", Z),
18         stdIO :: nl.
19
20     maximum(X, X, X).
21
22     maximum(X, Y, X) :-
23         X > Y.
24
25     maximum(X, Y, Y) :-
26         X < Y.
27
28 clauses
29     run() :-
30         console :: init(),
31         add(25, 11),
32         fadd(3.1223, 5.1111),
33         maximum(49, 5, Z),
34         stdIO :: write(Z),
35         fail.
36
37     run().
38
39 end implement main
40
41 goal
42     console :: runUtf8(main :: run).

```



```

Summary=36
FloatSummary=8.2334
49

```

Рис. 1.7: Результаты выполнения задания

## Задание 5

Программа находит страну, территория которой больше 1000000. Для использования анонимных переменных используется символ `_`.

```

1 implement main
2     open core
3
4 domains
5     название = symbol.
6     столица = symbol.
7     население = integer.
8     территория = real.
9
10 class predicates
11     страна : (название [out], население [out], территория [out], столица [out]) multi.
12 clauses
13     страна("kitai", 1200, 9597000, "pekin").

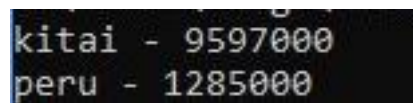
```



```

14     strana("belgia", 10, 30000, "brussel").
15     strana("peru", 20, 1285000, "lima").
16
17 clauses
18     run() :-
19         console::init(),
20         strana(X, _, Y, _),
21         Y > 1000000,
22         stdIO::writef("% - %\n", X, Y),
23         fail.
24
25     run().
26
27 end implement main
28
29 goal
30     console::runUtf8(main::run).

```



```

kitai - 9597000
peru - 1285000

```

Рис. 1.8: Результаты выполнения задания

## Задание 6

Используем собственный предикат `hello()` вместо стандартного `run()`.

```

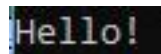
1 % Copyright
2
3 class main
4 open core
5
6 predicates
7 hello : ().
8
9 end class main

```

```

1 implement main
2     open core
3
4 clauses
5     hello() :-
6         console::init(),
7         stdIO::write("Hello!").
8
9 end implement main
10
11 goal
12     main::hello().

```



```

Hello!

```

Рис. 1.9: Результаты выполнения задания

## Задание 7

В ПРОЛОГе реализован механизм поиска с возвратом (backtracking), при котором система пытается отыскать все возможные решения задачи. Механизм вывода программы запоминает те точки процесса унификации, в которых не были использованы все альтернативные решения, а затем возвращается в эти точки и ищет решение по иному пути.

Для реализации данного механизма без взаимодействия с пользователем используется предикат `fail`.

```

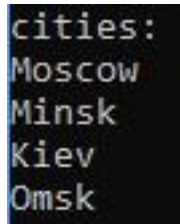
1 class main
2   open core
3
4   predicates
5   show : ().
6
7 end class main

```

```

1 implement main
2   open core
3
4   class predicates
5     gorod : (symbol [out]) multi.
6   clauses
7     gorod("Moscow").
8     gorod("Minsk").
9     gorod("Kiev").
10    gorod("Omsk").
11
12    show() :-
13      gorod(X),
14      stdIO::write(X),
15      stdIO::nl(),
16      fail.
17
18    show().
19
20 end implement main
21
22 goal
23   console::init(),
24   stdIO::write("cities:"),
25   stdIO::nl(),
26   main::show().

```



```

cities:
Moscow
Minsk
Kiev
Omsk

```

Рис. 1.10: Результаты выполнения задания

## Задание 8

Программа демонстрирует работу предиката `cut`. При нахождении хотя бы одного соответствия целям дальнейшие поиски прекращаются. В данном случае будут выведены имена всех мальчиков, девочки будут отброшены.

```

1 class main
2   open core
3
4   predicates
5   show : ().
6
7 end class main

```

```

1 implement main
2   open core
3

```

```

4 domains
5     person = symbol.
6
7 class predicates
8     deti : (person [out]) multi.
9     make_cut : (person) determ.
10
11 clauses
12     deti("Peter").
13     deti("Vasya").
14     deti("Oleg").
15     deti("Masha").
16     deti("Olyay").
17     deti("Natasha").
18
19     show() :-
20         deti(X),
21         stdIO :: write(X),
22         stdIO :: nl(),
23         make_cut(X),
24         !.
25
26     show().
27
28     make_cut(X) :-
29         X = "Oleg".
30
31 end implement main
32
33 goal
34     console :: init(),
35     stdIO :: write("Boys: "),
36     stdIO :: nl(),
37     main :: show().

```



```

Boys:
Peter
Vasya
Oleg

```

Рис. 1.11: Результаты выполнения задания

## Задание 9

В программе осуществляется поиск машины светлого цвета дешевле 25000. Программа не найдет ни одного решения, поскольку после дорогих зеленых «жигулей» поиск заканчивается, и более дешевые «ауди» не будут найдены.

```

1 implement main
2     open core
3
4 class predicates
5     buy_car : (symbol [out], symbol [out]) determ.
6     car : (symbol [out], symbol [out], integer [out]) multi.
7     color : (symbol, symbol) determ.
8
9 clauses
10     car("moskwich", "blue", 12000).
11     car("jiguli", "green", 26000).
12     car("volvo", "blue", 24000).
13     car("volga", "blue", 20000).
14     car("audi", "green", 20000).

```

```

15     color("blue", "dark").
16     color("green", "light").
17
18     buy_car(Model, Color) :-
19         car(Model, Color, Price),
20         color(Color, "light"),
21         !,
22         Price < 25000.
23
24     run() :-
25         console::init(),
26         buy_car(X, Y),
27         stdIO::writef("% - %\n", X, Y),
28         fail.
29
30     run().
31
32 end implement main
33
34 goal
35     console::runUtf8(main::run).

```

### Задание 10

В программе демонстрируется применение рекурсии. Будут напечатаны цифры от 1 до 9. В разделе clauses даны два описания предиката write\_number. Если в процессе решения первое описание не успешно, то используется второе описание.

```

1 implement main
2     open core
3
4 domains
5     number = integer.
6
7 class predicates
8     write_number : (number) nondeterm.
9 clauses
10    write_number(10).
11
12    write_number(N) :-
13        N < 10,
14        stdIO::write(N),
15        stdIO::nl(),
16        write_number(N + 1).
17
18    run() :-
19        console::init(),
20        stdIO::write("Numbers:"),
21        stdIO::nl(),
22        main::write_number(1),
23        fail.
24
25    run().
26
27 end implement main
28
29 goal
30     console::runUtf8(main::run).

```

```
Numbers:
1
2
3
4
5
6
7
8
9
```

Рис. 1.12: Результаты выполнения задания

## Задание 11

Данная программа печатает сумму всех цифр введенного числа. Использование предиката `!` в описании нерекурсивного правила позволяет избежать здесь переполнения стека.

```
1 implement main
2   open core
3
4 class predicates
5   summa : (integer, integer [out]).
6 clauses
7   summa(X, Y) :-
8     X < 10,
9     Y = X,
10    !.
11
12   summa(X, Y) :-
13     X1 = X div 10,
14     summa(X1, Y1),
15     Y = Y1 + X mod 10.
16
17   run() :-
18     console::init(),
19     summa(138965, Y),
20     stdIO::write(Y),
21     stdIO::nl().
22
23 end implement main
24
25 goal
26   console::runUtf8(main::run).
```



Рис. 1.13: Результаты выполнения задания

## Задание 12

Данная программа решает задачу «Ханойская башня». Требуется переместить диски с первого на третий стержень за некоторую последовательность ходов, каждый из которых заключается в перекладывания верхнего диска с одного из стержней на другой стержень. При этом больший диск никогда нельзя ставить на меньший диск.

В результате работы программы получено описание действий, необходимых для решения данной задачи.

```
1 implement main
2   open core
3
4 domains
5   loc = right; middle; left.
6
7 class predicates
8   hanoi : (integer).
9   move : (integer, loc, loc, loc).
10  inform : (loc, loc).
11
12 clauses
13   hanoi(N) :-
14     move(N, left, middle, right).
15
16   move(1, A, _, C) :-
17     inform(A, C),
18     !.
19
```

```
20     move(N, A, B, C) :-
21         move(N - 1, A, C, B),
22         inform(A, C),
23         move(N - 1, B, A, C).
24
25     inform(Loc1, Loc2) :-
26         stdIO::write("Disk C ", Loc1, " in ", Loc2),
27         stdIO::nl().
28
29     run() :-
30         console::init(),
31         hanoi(5).
32
33 end implement main
34
35 goal
36     console::runUtf8(main::run).
```

```
Disk C left in right
Disk C left in middle
Disk C right in middle
Disk C left in right
Disk C middle in left
Disk C middle in right
Disk C left in right
Disk C left in middle
Disk C right in middle
Disk C right in left
Disk C middle in left
Disk C right in middle
Disk C left in right
Disk C left in middle
Disk C right in middle
Disk C left in right
Disk C middle in left
Disk C middle in right
Disk C left in right
Disk C middle in left
Disk C right in middle
Disk C right in left
Disk C middle in left
Disk C middle in right
Disk C left in right
Disk C left in middle
Disk C right in middle
Disk C left in right
Disk C middle in left
Disk C middle in right
Disk C left in right
```

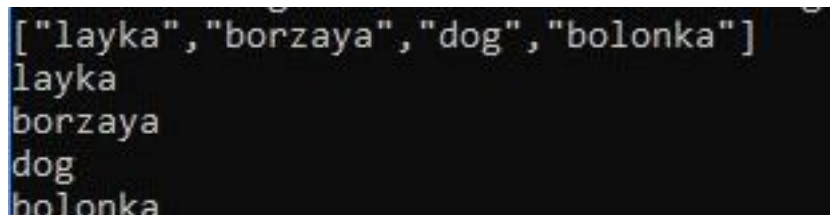
Рис. 1.14: Результаты выполнения задания



### Задание 13

В данной программе используются списки. С их помощью описываются породы собак. Операция разделения списка на голову и хвост обозначается с помощью вертикальной черты: [Head|Tail]. С помощью этой операции можно реализовывать рекурсивную обработку списка и вывести элементы списка построчно.

```
1 implement main
2   open core
3
4 domains
5   dog_list = symbol*.
6
7 class predicates
8   dogs : (dog_list [out]).
9   print_list : (dog_list).
10
11 clauses
12   dogs(["layka", "borzaya", "dog", "bolonka"]).
13
14   print_list([]).
15
16   print_list([X | Y]) :-
17     stdIO::write(X),
18     stdIO::nl(),
19     print_list(Y).
20
21   run() :-
22     console::init(),
23     dogs(X),
24     stdIO::write(X),
25     stdIO::nl(),
26     print_list(X).
27
28 end implement main
29
30 goal
31   console::runUtf8(main::run).
```



```
["layka","borzaya","dog","bolonka"]
layka
borzaya
dog
bolonka
```

Рис. 1.15: Результаты выполнения задания

### Задание 14

Первое правило описывает ситуацию, когда искомый элемент X совпадает с головой списка. Второе правило используется при неуспехе первого правила и описывает новый вызов первого правила, но уже с усеченным списком, в котором нет первого элемента и т.д. Если в списке нет элементов (пустой список), то второе правило оказывается неуспешным.

Программа не напечатает Yes, поскольку болонки нет в списке собак.

```
1 implement main
2   open core
3
4 domains
5   dog_list = symbol*.
6
7 class predicates
8   find_it : (symbol, dog_list) nondeterm.
9 clauses
```

```

10 find_it(X, [X | _]).
11
12 find_it(X, [_ | Y]) :-
13     find_it(X, Y).
14
15 run() :-
16     console::init(),
17     find_it("bolonka", ["layka", "dog"]),
18     stdIO::write("yes"),
19     fail.
20
21 run().
22
23 end implement main
24
25 goal
26     console::runUtf8(main::run).

```

### Задание 15

В данном примере производится подсчет суммы всех элементов списка.

```

1 implement main
2     open core
3
4 domains
5     spisok = integer*.
6
7 class predicates
8     summa_sp : (spisok, integer [out]).
9 clauses
10    summa_sp([], 0).
11
12    summa_sp([H | T], S) :-
13        summa_sp(T, S1),
14        S = H + S1.
15
16    run() :-
17        console::init(),
18        summa_sp([4, 5, 0, 1, 9], Sum),
19        stdIO::write(Sum),
20        stdIO::nl().
21
22 end implement main
23
24 goal
25     console::runUtf8(main::run).

```

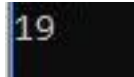


Рис. 1.16: Результаты выполнения задания

### Задание 16

Данный пример показывает, как используются списки и механизм рекурсии при решении известной задачи о мужике, волке, козе и капусте.

```

1 implement main
2     open core
3
4 domains
5     loc = east; west.

```

```

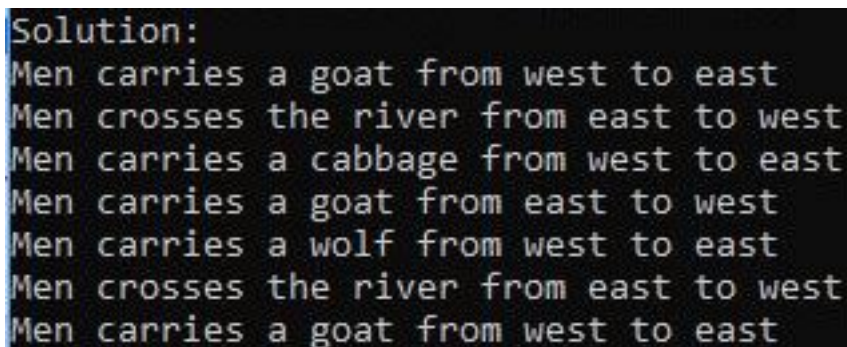
6      state = state(loc, loc, loc, loc).
7      path = state*.
8
9  class predicates
10     go : (state, state).
11     path : (state, state, path, path [out]) determ.
12     move : (state, state [out]) nondeterm.
13     opposite : (loc, loc) determ anyflow.
14     unsafe : (state) nondeterm.
15     member : (state, path) nondeterm.
16     write_path : (path) determ.
17     write_move : (state, state) determ.
18
19  clauses
20     go(S, G) :-
21         path(S, G, [S], L),
22         stdIO::write("Solution:"),
23         stdIO::nl(),
24         write_path(L),
25         fail.
26
27     go(_, _).
28
29     path(S, G, L, L1) :-
30         move(S, S1),
31         not(unsafe(S1)),
32         not(member(S1, L)),
33         path(S1, G, [S1 | L], L1),
34         !.
35
36     path(G, G, T, T) :-
37         !.
38
39     move(state(X, X, G, C), state(Y, Y, G, C)) :-
40         opposite(X, Y).
41
42     move(state(X, W, X, C), state(Y, W, Y, C)) :-
43         opposite(X, Y).
44
45     move(state(X, W, G, X), state(Y, W, G, Y)) :-
46         opposite(X, Y).
47
48     move(state(X, W, G, C), state(Y, W, G, C)) :-
49         opposite(X, Y).
50
51     opposite(east, west).
52
53     opposite(west, east) :-
54         !.
55
56     unsafe(state(F, X, X, _)) :-
57         opposite(F, X).
58
59     unsafe(state(F, _, X, X)) :-
60         opposite(F, X).
61
62     member(X, [X | _]).
63
64     member(X, [_ | L]) :-
65         member(X, L).
66
67     write_path([H1, H2 | T]) :-
68         !,
69         write_move(H1, H2),
70         write_path([H2 | T]).
71

```

```

72 write_path([]).
73
74 write_move(state(X, W, G, C), state(Y, W, G, C)) :-
75     !,
76     stdIO::write("Men crosses the river from ", X, " to ", Y),
77     stdIO::nl().
78
79 write_move(state(X, X, G, C), state(Y, Y, G, C)) :-
80     !,
81     stdIO::write("Men carries a wolf from ", X, " to ", Y),
82     stdIO::nl().
83
84 write_move(state(X, W, X, C), state(Y, W, Y, C)) :-
85     !,
86     stdIO::write("Men carries a goat from ", X, " to ", Y),
87     stdIO::nl().
88
89 write_move(state(X, W, G, X), state(Y, W, G, Y)) :-
90     !,
91     stdIO::write("Men carries a cabbage from ", X, " to ", Y),
92     stdIO::nl().
93
94 clauses
95     run() :-
96         console::init(),
97         go(state(east, east, east, east), state(west, west, west, west)).
98
99 end implement main
100
101 goal
102     console::runUtf8(main::run).

```



```

Solution:
Men carries a goat from west to east
Men crosses the river from east to west
Men carries a cabbage from west to east
Men carries a goat from east to west
Men carries a wolf from west to east
Men crosses the river from east to west
Men carries a goat from west to east

```

Рис. 1.17: Результаты выполнения задания

### Задание 17

В данной программе решается следующая логическая задача:

«В велосипедных гонках три первых места заняли Алеша, Петя и Коля. Какое место занял каждый из них, если Петя занял не второе и не третье место, а Коля – не третье?»

```

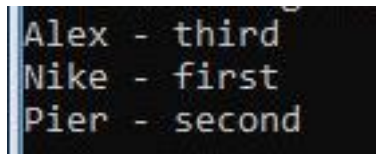
1 implement main
2     open core
3
4 class predicates
5     name : (symbol) determ.
6     name : (symbol [out]) multi.
7     mesto : (symbol) determ.
8     mesto : (symbol [out]) multi.
9     prizer : (symbol, symbol) nondeterm.
10    solution : (symbol [out], symbol [out], symbol [out], symbol [out], symbol [out],
11               symbol [out]) determ.

```

```

12 clauses
13     name("Alex").
14     name("Pier").
15     name("Nike").
16     mesto("first").
17     mesto("second").
18     mesto("third").
19
20     prizer(X, Y) :-
21         name(X),
22         mesto(Y),
23         X = "Pier",
24         not(Y = "second"),
25         not(Y = "third")
26     or
27         name(X),
28         mesto(Y),
29         X = "Nike",
30         not(Y = "third")
31     or
32         name(X),
33         mesto(Y),
34         not(X = "Pier"),
35         not(X = "Nike").
36
37     solution(X1, Y1, X2, Y2, X3, Y3) :-
38         name(X1),
39         name(X2),
40         name(X3),
41         mesto(Y1),
42         mesto(Y2),
43         mesto(Y3),
44         prizer(X1, Y1),
45         prizer(X2, Y2),
46         prizer(X2, Y3),
47         Y1 <> Y2,
48         Y2 <> Y3,
49         Y1 <> Y3,
50         X1 <> X2,
51         X2 <> X3,
52         X1 <> X3,
53         !.
54
55     run() :-
56         console::init(),
57         solution(X1, Y1, X2, Y2, X3, Y3),
58         stdIO::writef("% - %\n% - %\n% - %\n", X1, Y1, X2, Y2, X3, Y3),
59         fail.
60
61     run().
62
63 end implement main
64
65 goal
66     console::runUtf8(main::run).

```



```

Alex - third
Nike - first
Pier - second

```

Рис. 1.18: Результаты выполнения задания

## Задание 18

В данной программе решается следующая логическая задача:

«Пятеро студентов едут на велосипедах. Их зовут Сергей, Борис, Леонид, Григорий и Виктор. Велосипеды сделаны в пяти городах: Риге, Пензе, Львове, Харькове и Москве. Каждый из студентов родился в одном из этих городов, но ни один из студентов не едет на велосипеде, сделанном на его родине. Сергей едет на велосипеде, сделанном в Риге. Борис родом из Риги, у него велосипед из Пензы. У Виктора велосипед из Москвы. У Григория велосипед из Харькова. Виктор родом из Львова. Уроженец Пензы едет на велосипеде, сделанном на родине Леонида. Кто из студентов родом из Москвы?»

```
1 implement main
2     open core
3
4 domains
5     name = symbol.
6
7 class predicates
8     student : (name) determ.
9     student : (name [out]) multi.
10    gorod : (name) determ.
11    gorod : (name [out]) multi.
12    velo : (name, name) determ.
13    fact : (name, name) determ anyflow.
14    fact1 : (name, name) determ anyflow.
15    rodом : (name, name) nondeterm.
16    rodом : (name [out], name) nondeterm.
17    rodом_penza : (name) nondeterm.
18
19 clauses
20     student(X) :-
21         X = "Sergey"
22         or
23         X = "Boris"
24         or
25         X = "Viktor"
26         or
27         X = "Gregory"
28         or
29         X = "Leonid".
30
31     gorod(Y) :-
32         Y = "Penza"
33         or
34         Y = "L'vov"
35         or
36         Y = "Moscow"
37         or
38         Y = "Har'kov"
39         or
40         Y = "Riga".
41
42     fact("Sergey", "Riga").
43     fact("Boris", "Penza").
44     fact("Viktor", "Moscow").
45     fact("Gregory", "Har'kov").
46
47     velo(X, Y) :-
48         student(X),
49         gorod(Y),
50         fact(X, Y),
51         !
52         or
53         student(X),
54         gorod(Y),
55         not(fact(X, _)),
56         not(fact(_, Y)).
```

```

57
58 fact1("Boris", "Riga").
59 fact1("Viktor", "L'vov").
60
61 rodom_penza(X) :-
62     student(X),
63     not(fact1(X, _)),
64     gorod(U),
65     not(U = "Penza"),
66     velo(X, U),
67     rodom("Leonid", U).
68
69 rodom(X, Z) :-
70     student(X),
71     gorod(Z),
72     fact1(X, Z),
73     !
74     or
75     student(X),
76     not(X = "Leonid"),
77     Z = "Penza",
78     rodom_penza(X),
79     !
80     or
81     student(X),
82     gorod(Z),
83     not(fact1(_, Z)),
84     X = "Leonid",
85     not(Z = "Penza"),
86     student(K),
87     not(fact1(K, _)),
88     velo(K, Z)
89     or
90     student(X),
91     not(X = "Leonid"),
92     gorod(Z),
93     not(Z = "Penza"),
94     not(fact1(_, Z)),
95     not(fact1(X, _)),
96     gorod(Y),
97     not(Y = Z),
98     velo(X, Y),
99     not(rodom("Leonid", Z)),
100    not(rodom("Leonid", Y)).
101
102 run() :-
103     console::init(),
104     rodom(X, "Moscow"),
105     stdIO::writef("% from Moscow", X),
106     fail.
107
108 run().
109
110 end implement main
111
112 goal
113     console::runUtf8(main::run).

```



Рис. 1.19: Результаты выполнения задания

## Задание 19

В данной программе решается следующая логическая задача:

«Пять студентов должны посещать лекции всю неделю, но по определенным ими установленным правилам, а именно: 1. Если пришли Андрей и Дмитрий, то Бориса быть не должно, но если Дмитрий не пришел, то Борис должен быть, а Виктор быть не должен. 2. Если Виктор пришел, то Андрея быть не должно и наоборот. 3. Если Дмитрий пришел, то Григория быть не должно. 4. Если Бориса нет, то Дмитрий должен быть, но если нет также и Виктора, а если Виктор есть, Дмитрия быть не должно, но должен быть Григорий. 5. Каждый день студенты должны приходить в разных сочетаниях. Какие это сочетания?»

```
1 implement main
2     open core
3
4 domains
5     s = symbol.
6
7 class predicates
8     st_A : (s [out]) multi.
9     st_D : (s [out]) multi.
10    st_B : (s [out]) multi.
11    st_V : (s [out]) multi.
12    st_G : (s [out]) multi.
13    ogr1 : (s, s, s, s, s) determ.
14    ogr2 : (s, s, s, s, s) determ.
15    spisok : (s [out], s [out], s [out], s [out], s [out]) nondeterm.
16    norm1 : (s, s, s, s, s) determ.
17    norm2 : (s, s, s, s, s) determ.
18    norm3 : (s, s, s, s, s) determ.
19    norm4 : (s, s, s, s, s) determ.
20
21 clauses
22     st_A(A) :-
23         A = "Andrey"
24         or
25         A = ",no".
26
27     st_D(D) :-
28         D = "Dmitry"
29         or
30         D = ",no".
31
32     st_B(B) :-
33         B = "Boris"
34         or
35         B = ",no".
36
37     st_V(V) :-
38         V = "Viktor"
39         or
40         V = ",no".
41
42     st_G(G) :-
43         G = "Gregory"
44         or
45         G = ",no".
46
47     ogr1("Andrey", _, _, ",no", _).
48     ogr1(",no", _, _, "Viktor", _).
49     ogr2(_, "Dmitry", _, _, ",no").
50     ogr2(_, ",no", _, _, _).
51     norm1("Andrey", "Dmitry", ",no", _, _).
52     norm2("Andrey", ",no", "Boris", ",no", _).
53     norm3(_, "Dmitry", ",no", ",no", _).
54     norm4(_, ",no", ",no", "Viktor", "Gregory").
55
56     spisok(A, D, B, V, G) :-
```



```

57     st_A(A),
58     st_D(D),
59     st_B(B),
60     st_V(V),
61     st_G(G),
62     norm1(A, D, B, V, G),
63     ogr1(A, D, B, V, G),
64     ogr2(A, D, B, V, G)
65     or
66     st_A(A),
67     st_D(D),
68     st_B(B),
69     st_V(V),
70     st_G(G),
71     norm2(A, D, B, V, G),
72     ogr1(A, D, B, V, G),
73     ogr2(A, D, B, V, G)
74     or
75     st_A(A),
76     st_D(D),
77     st_B(B),
78     st_V(V),
79     st_G(G),
80     norm3(A, D, B, V, G),
81     ogr1(A, D, B, V, G),
82     ogr2(A, D, B, V, G)
83     or
84     st_A(A),
85     st_D(D),
86     st_B(B),
87     st_V(V),
88     st_G(G),
89     norm4(A, D, B, V, G),
90     ogr1(A, D, B, V, G),
91     ogr2(A, D, B, V, G)
92     or
93     st_A(A),
94     st_D(D),
95     st_B(B),
96     st_V(V),
97     st_G(G),
98     not(norm1(A, D, B, V, G)),
99     not(norm2(A, D, B, V, G)),
100    not(norm3(A, D, B, V, G)),
101    not(norm4(A, D, B, V, G)),
102    ogr1(A, D, B, V, G),
103    ogr2(A, D, B, V, G).
104
105 clauses
106     run() :-
107         console::init(),
108         spisok(A, D, B, V, G),
109         stdIO::writef("% % % % %\n", A, D, B, V, G),
110         fail.
111
112     run().
113
114 end implement main
115
116 goal
117     console::runUtf8(main::run).

```

```

Andrey Dmitry no, no, no,
Andrey no, Boris no, Gregory
Andrey no, Boris no, no,
Andrey Dmitry no, no, no,
no, no, no, Viktor Gregory
Andrey Dmitry Boris no, no,
Andrey no, no, no, Gregory
Andrey no, no, no, no,
no, Dmitry Boris Viktor no,
no, Dmitry no, Viktor no,
no, no, Boris Viktor Gregory
no, no, Boris Viktor no,
no, no, no, Viktor no,

```

Рис. 1.20: Результаты выполнения задания

## Задание 20

Факты, описанные в разделе `clauses`, можно рассматривать, как статическую базу данных (БД). Эти факты являются частью кода программы и не могут быть оперативно изменены.

В данной программе осуществляется поиск человека, ростом выше 180 см.

```

1 implement main
2     open core
3
4 domains
5     name = symbol.
6     rost = integer.
7     ves = integer.
8
9 class facts
10     dplayer : (name, rost, ves).
11
12 class predicates
13     player : (name [out], rost [out], ves [out]) multi.
14     assert_database : ().
15
16 clauses
17     player("M", 180, 87).
18     player("P", 187, 93).
19     player("H", 177, 80).
20
21     assert_database() :-
22         player(N, R, V),
23         assertz(dplayer(N, R, V)),
24         fail.
25
26     assert_database().
27
28 clauses
29     run() :-
30         console::init(),
31         assert_database(),
32         dplayer(N, R, V),
33         R > 180,
34         stdIO::writef("% % sm % kg", N, R, V),
35         fail.
36
37     run().
38

```

```

39 end implement main
40
41 goal
42   console :: runUtf8 (main :: run) .

```

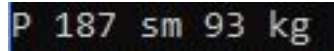


Рис. 1.21: Результаты выполнения задания

## Задание 21

Создадим простую экспертную систему, которая решает задачу определения вида экземпляра пойманной рыбы.

Программа реализует заданное дерево поиска решения. Ответы на заданные вопросы позволяют продвигаться по ветвям этого дерева к одному из вариантов решения.

```

1  implement main
2      open core
3
4  class facts
5      xpositive : (string , string) .
6      xnegative : (string , string) .
7
8  class predicates
9      expertiza : () .
10     vopros : (string , string) determ .
11     fish_is : (string [out]) nondeterm .
12     positive : (string , string) determ .
13     negative : (string , string) determ .
14     remember : (string , string , string) determ .
15     clear_facts : () .
16
17  clauses
18     expertiza () :-
19         fish_is (X) ,
20         ! ,
21         stdIO :: write ("u fish is " , X , " ") ,
22         stdIO :: nl ,
23         clear_facts () .
24
25     expertiza () :-
26         stdIO :: write ("noname!") ,
27         stdIO :: nl ,
28         clear_facts () .
29
30     vopros (X , Y) :-
31         stdIO :: write ("question - " , X , " " , Y , "? (y/n)") ,
32         R = stdIO :: readLine () ,
33         remember (X , Y , R) .
34
35     positive (X , Y) :-
36         xpositive (X , Y) ,
37         ! .
38
39     positive (X , Y) :-
40         not (negative (X , Y)) ,
41         ! ,
42         vopros (X , Y) .
43
44     negative (X , Y) :-
45         xnegative (X , Y) ,
46         ! .
47
48     remember (X , Y , "y") :-

```

```

49      assertz(xpositive(X, Y)).
50
51  remember(X, Y, "n") :-
52      assertz(xnegative(X, Y)),
53      fail.
54
55  clear_facts() :-
56      retract(xpositive(_, _)),
57      fail.
58
59  clear_facts() :-
60      retract(xnegative(_, _)),
61      fail.
62
63  clear_facts().
64
65  fish_is("som") :-
66      positive("fish have ", "weight > 40 kg").
67
68  fish_is("som") :-
69      positive("fish have ", "weight < 40 kg"),
70      positive("fish have ", " mustache").
71
72  fish_is("shchuka") :-
73      positive("fish have ", "weight < 20 kg"),
74      positive("fish have ", "long narrow body").
75
76  fish_is("perch") :-
77      positive("fish have ", "weight < 20 kg"),
78      positive("fish have ", "wide body"),
79      positive("fish have ", "dark hair").
80
81  fish_is("roach") :-
82      positive("fish have ", "weight < 20 kg"),
83      positive("fish have ", "wide body"),
84      positive("fish have ", "gray scales").
85
86  clauses
87      run() :-
88          console::init(),
89          expertiza().
90
91  end implement main
92
93  goal
94      console::run(main::run).

```

```

question - fish have weight > 40 kg? (y/n)n
question - fish have weight < 40 kg? (y/n)n
question - fish have weight < 20 kg? (y/n)y
question - fish have long narrow body? (y/n)y
u fish is shchuka

```

Рис. 1.22: Результаты выполнения задания

### 1.3.6 Выполнить индивидуальное задание из пособия Буракова С.В.

Витя, Юра и Миша сидели на скамейке. В каком порядке они сидели, если известно, что Юра сидел слева от Миши и справа от Вити.

```

1  implement main
2      open core

```

```

3
4 domains
5     person = symbol.
6
7 class predicates
8     left : (person, person) nondeterm anyflow.
9     pos : (person [out], person [out], person [out]) nondeterm.
10
11 clauses
12
13     left("Vitya", "Ura").
14     left("Ura", "Misha").
15
16     pos(X, Y, Z) :-
17         left(X, Y),
18         left(Y, Z).
19
20 clauses
21     run() :-
22         console::init(),
23         pos(X, Y, Z),
24         stdIO::writef(X, " - ", Y, " - ", Z),
25         fail.
26
27     run().
28
29 end implement main
30
31 goal
32     console::runUtf8(main::run).

```

Результат выполнения задания:

```
Vitya - Ura - Misha
```

Рис. 1.23: Результаты выполнения индивидуального задания

### 1.3.7 Выполнить индивидуальное задание из пособия Седана С.Н.

Лабиринт представляет собой систему комнат, соединенных между собой переходами. В лабиринте имеется вход и выход, а также комната с золотым кладом. Кроме того, имеются комнаты, запрещенные для посещений: комната монстров и комната разбойников.

1. Найди путь в лабиринте от входа до выхода, не посещая дважды одной и той же комнаты;
2. Найти путь с посещением золотой комнаты;
3. Найти путь, избегающий запрещенных к посещению комнат.

```

1 implement main
2     open core
3
4 class predicates
5     connection : (string [out], string [out]) multi.
6     connection : (string, string [out]) nondeterm.
7     connection : (string [out], string) nondeterm.
8     path : (string, string, string*) nondeterm.
9     member : (string, string*) nondeterm.
10    reverse : (string*, string* [out], string*) nondeterm anyflow.
11
12 clauses
13     connection("entrance", "room 1").
14
15     connection("entrance", "room 4").

```

```

16 connection("room 1", "gold").
17
18 connection("room 1", "robbers").
19
20 connection("room 1", "room2").
21
22 connection("room2", "exit").
23
24 connection("room 3", "exit").
25
26 connection("room 4", "room2").
27
28 connection("gold", "monster").
29
30 connection("gold", "room 3").
31
32 connection("monster", "exit").
33
34 connection("robbers", "exit").
35
36
37 member(X, [Y | T]) :-
38     X = Y
39     or
40     member(X, T).
41
42 reverse([], Z, Z).
43
44 reverse([H | T], Z, A) :-
45     reverse(T, Z, [H | A]).
46
47 path(Y, Y, _).
48
49 path(X, Y, T) :-
50     connection(X, Z),
51     not(member(Z, T)),
52     path(Z, Y, [Z | T]).
53
54 path(X, Y, T) :-
55     connection(Z, X),
56     not(member(Z, T)),
57     path(Z, Y, [Z | T]).
58
59 path(_, _, [F | O]) :-
60     reverse([F | O], [L | V], []),
61     if F = "exit" and L = "entrance" and not(member("robbers", O)) and not(member("
monster", O)) and member("gold", O) then
62         stdIO::writef("%\n", [L | V])
63     else
64         end if.
65
66 run() :-
67     console::init(),
68     connection(X, Y),
69     path(X, Y, []),
70     fail.
71
72 run().
73
74 end implement main
75
76 goal
77     console::runUtf8(main::run).

```

В данном алгоритме указываются связи между соседними комнатами, после чего подбираются различные варианты маршрутов, с помощью path. Кроме того, на path установлен фильтр, который выводит в консоль

все варианты пути, удовлетворяющие фильтру. Таким образом выполняются условия с типами комнат. Результат выполнения задания:

```
["entrance", "room 1", "gold", "room 3", "exit"]
["entrance", "room 4", "room2", "room 1", "gold", "room 3", "exit"]
["entrance", "room 1", "gold", "room 3", "exit"]
["entrance", "room 4", "room2", "room 1", "gold", "room 3", "exit"]
["entrance", "room 1", "gold", "room 3", "exit"]
["entrance", "room 4", "room2", "room 1", "gold", "room 3", "exit"]
["entrance", "room 1", "gold", "room 3", "exit"]
["entrance", "room 4", "room2", "room 1", "gold", "room 3", "exit"]
```

Рис. 1.24: Результаты выполнения индивидуального задания

Варианты правильные, однако, повторяются несколько раз. Этого можно избежать, если заменить `writeln` внутри функции на возвращаемый список, который впоследствии будет отфильтрован на предмет одинаковых вариантов.

## 1.4 Вывод

В результате работы были изучены основные возможности языка Prolog, а также среды разработки Visual Prolog.

### Плюсы и минусы языка Prolog

К плюсам языка Prolog относится тот факт, что это декларативный язык: описывается логическая модель предметной области (ПО) в терминах этой ПО, их свойства и отношения между ними, а не детали программной реализации. Таким образом, мы описываем «что» хотим получить, а не «как» мы хотим это получить. Это позволяет сильно уменьшить время разработки приложений и размеры исходного кода.

Из минусов этого языка можно выделить его узкую специализацию. Язык хорошо показывает себя в своей области применения, однако, для написания чего-то универсального и производительного он не годится.

### Аналоги языка Prolog

Существует множество аналогов языка PROLOG, например:

- Python – язык, который содержит огромный набор научных библиотек, в том числе и для ИИ.
- CLIPS – программная среда и язык для разработки экспертных систем.
- Planner – функционально-логический язык программирования, схожий по своему синтаксису с LISP.
- Linden Scripting Language (или LSL) – скриптовый язык программирования.
- LISP – функциональный язык программирования.

### Проблема комбинаторного взрыва

Проблема комбинаторного взрыва в общем случае не решена, однако, использование метода наискорейшего подъема помогает значительно ускорить поиск.

### Корректность разработки языка ИИ

Идея разработки языка ИИ не только корректна, но и необходима. Кроме того, существующие языки в этой области должны быть или улучшены, или заменены более совершенными и удобными языками. Это позволит значительно ускорить создание полноценного ИИ. Разумеется, обязательным качеством такого языка должна являться декларативность.

### Понимание ИИ

Понимание о идеальном ИИ должно быть поверхностным, на уровне основных высокоуровневых компонентов и ограничений. Все остальное нужно предоставить самому ИИ на самостоятельное обучение. Это также соответствует декларативной концепции – мы должны знать что делает ИИ, но не должны знать как.

## 1.5 Список литературы

- [illegible]