Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №4(Язык искусственного интеллекта PROLOG)**

**Дисциплина**: Интеллектуальные системы

Выполнил студент гр. 13541/3 С. В. Смирнов

(подпись)

Преподаватель Е. Н. Бендерская

(подпись)

Санкт-Петербург

2017

Оглавление

[Задание: 3](#_Toc497159970)

[Решение 4](#_Toc497159971)

[Вывод: 40](#_Toc497159973)

[Список используемой литературы 42](#_Toc497159974)

Задание:

1 Получите начальное представление о синтаксисе и семантике базовых конструкций языка PROLOG, ознакомившись с разделами 1-5 методического пособия: Бураков С. В. «Язык логического программирования PROLOG», СПбГУАП, 2003.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА, НО ОЧЕНЬ ПОЛЕЗНАЯ (!)

Середа С.Н. «Методичка по языку Prolog», Муромский университет. 2003г.

2 Создайте проект в оболочке Visual Prolog 7.3., как это показано в примере

http://wikiru.visual-рrolog.com/index.php?title=%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%

D1%8B\_%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B\_Visual\_Prolog

3 Удалите проект, созданный в п. 2 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

(подсказка: используйте понятие фрейма).

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. При

создании проектов указывать тип «консольный». Чтобы протестировать консольную

программу, используйте команду Run in Window, не Execute.

8 Изучить 1-2 лабы по методичке Седана С.Н. (доп литература).Согласно своему варианту решить задачу с помощью PROLOG .

9. Написать выводы. В выводах отразить, помимо своих мыслей, возникших в ходе работы, ответы на приведенные ниже вопросы:

1. В чем Плюсы и минусы языка Prolog?

2. Какие еще языки используются для разработки ИИ, приведите примеры (НЕ МЕНЕЕ 2-х) проектов, языков и краткое описание проектов. (Альтернативы PROLOG)

3. Решаема ли проблема комбинаторного взрыва, пути решения?

4. Корректно ли по-вашему в принципе разработка языка ИИ? Что он должен из себя представлять?

5. Можно ли разработать ИИ не понимая, как он работает, должны ли мы понимать, как он работает, думает, рассуждает?

Решение**:**

1. Ознакомился с литературой п.1. Был создан проект по примеру из п.2, удалён и запущен демонстрационный проект family1.[1]
2. По результатам работы программы было создано следующее генеалогическое дерево (выше – предки, ниже - потомки):

Judith

|

John

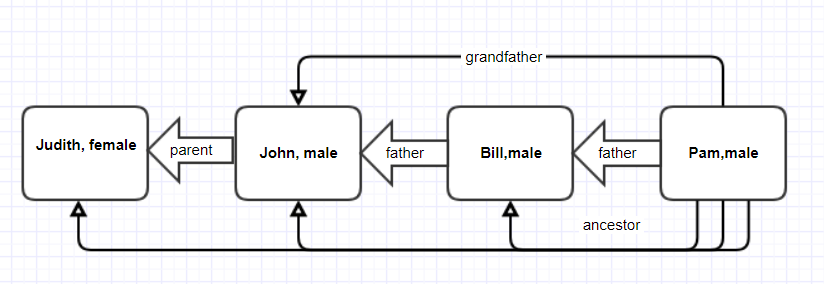
|

Bill

|

Pam

1. Требуется определить степень родства группы людей, для каждого из которых определены следующие параметры: имя, пол, родитель(не для всех указан). Первоначально определяем отцовство( персона мужского пола и являющаяся родителем). Затем определяется наличие внуков( персона является родителем родителя и мужского пола). После чего определяются предки(ancestor) для внука (Рam). Исходя из полученной информации возможно построение генеалогического дерева данной семьи.
2. Построение концептуальной карты для рассматриваемого примера

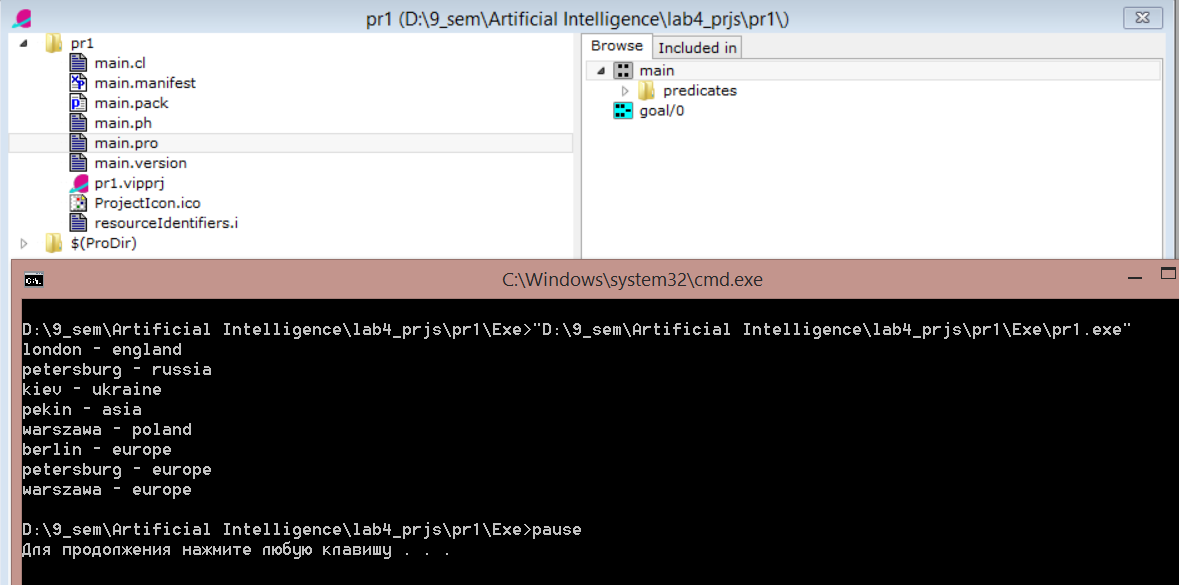


1. Создание проектов 1-21 по пособию

Программа 1.

implement main  
    open core  
      
    class predicates  
        situ : (string Gorod, string Strana) nondeterm anyflow.  
    clauses  
            situ ("london", "england").  
            situ ("petersburg", "russia").  
            situ ("kiev", "ukraine").  
            situ ("pekin", "asia").  
            situ ("warszawa", "poland").  
            situ ("berlin", "europe").  
            situ (X, "europe"):-  
                situ (X, "russia").  
            situ (X, "europe"):-  
                situ (X, "poland").  
      
    clauses  
        run():-  
            console::init(),  
            situ(X, Y),  
            stdIO::writef("% - %\n", X, Y),  
            fail.  
        run().  
    end implement main  
      
    goal  
        mainExe::run(main::run).

Результат выполнения:

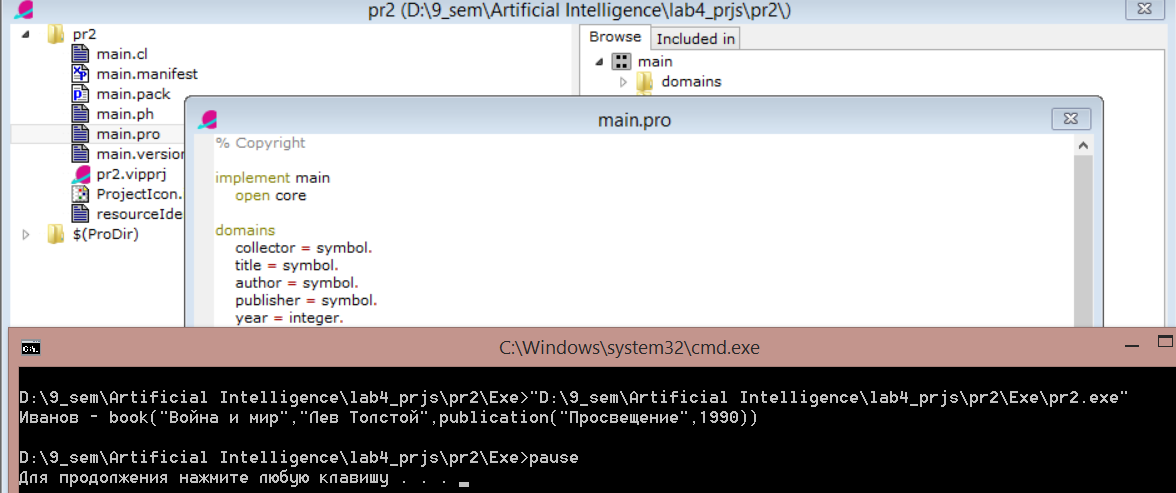


Данная программа является простой демонстрацией синтаксиса языка и выводит всю информацию ассоциированную с suty.

Программа 2.

implement main  
    open core  
  
domains  
    collector=symbol.  
    title=symbol.  
    author=symbol.  
    publisher=symbol.  
    year = integer.  
    personal\_library = book(title, author, publication).  
    publication = publication(publisher, year).  
  
class predicates  
    collection : (collector[out], personal\_library[out]).  
      
clauses  
    collection("Иванов", book("Война и мир", "Лев Толстой", publication("Просвещение",1990))).  
          
clauses  
    run():-  
        console::init(),  
        collection(X, Y),  
        stdIO::writef("% - %\n", X, Y),  
        fail.  
    run().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

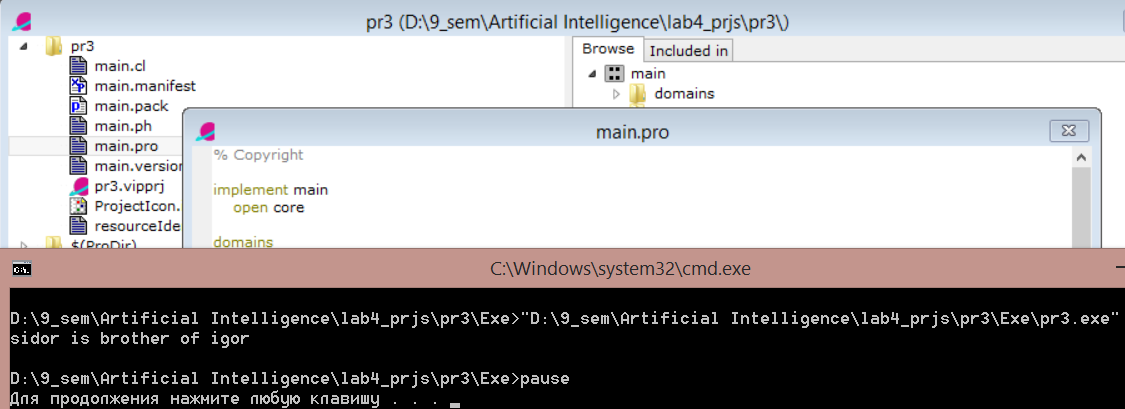


Данная программа показывает, как в ПРОЛОГе можно использовать составные объекты. Составные объекты позволяют описывать иерархические структуры, в которых описание одного предиката включает в себя описание других предикатов.

Программа 3.

implement main  
    open core  
  
domains  
    person = symbol.  
class predicates  
    otec : (person, person) nondeterm anyflow.  
    man : (person) nondeterm.  
    brat : (person[out], person[out]) nondeterm.  
clauses  
    man(X):-  
        otec(X,\_).  
    brat(X,Y):-  
        otec(Z,Y),  
        otec(Z,X),  
        man(X),  
        X<>Y.  
    otec("ivan","igor").   
    otec("ivan","sidor").   
    otec("sidor","lisa").  
  
clauses  
    run():-  
        console::init(),  
        brat(X, Y),  
        stdIO::writef("% is brother of %\n", X, Y),  
        fail.  
    run().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

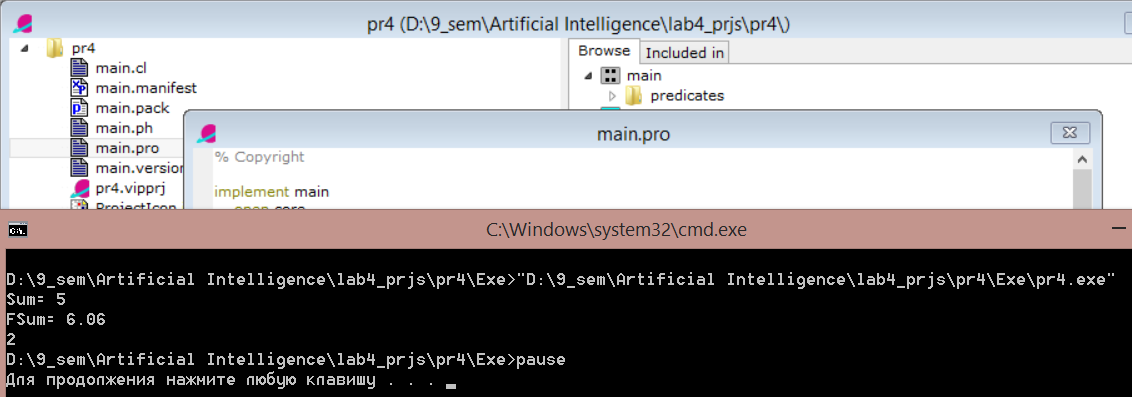


Программа решает примитивную задачу о семейных отношениях, по сходству с family1. В результате система находит брата Игоря – Сидора по имеющимся входным данным.

Программа 4.

implement main  
    open core  
class predicates  
    add : (integer, integer).  
    fadd : (real, real).  
    maximum : (real, real, real[out]) nondeterm.  
clauses  
    add(X, Y):-  
        Z = X + Y,  
        stdIO::write("Sum= ", Z),  
        stdIO::nl.  
    fadd(X, Y):-  
        Z = X + Y,  
        stdIO::write("FSum= ", Z),  
        stdIO::nl.  
    maximum(X, X, X).  
    maximum(X, Y, X):-  
        X>Y.  
    maximum(X, Y, Y):-  
        X<Y.  
clauses  
    run():-  
        console::init(),  
        add(2, 3),  
        fadd(3.5, 2.56),  
        maximum(2, -4, Z),  
        stdIO::write(Z),  
        fail.  
    run().  
end implement main  
goal  
    mainExe::run(main::run).

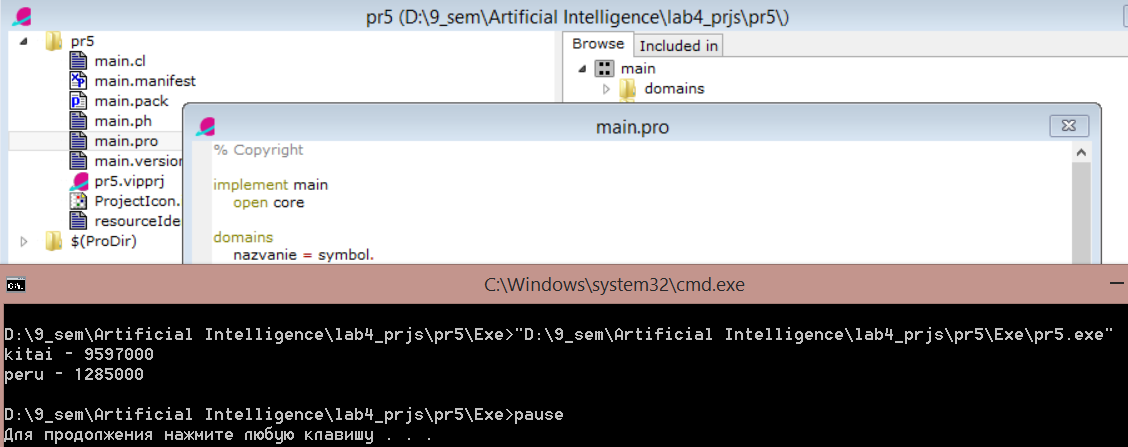
Результат работы:



В данной задаче находится сумма целых чисел, сумма вещественных чисел и максимум из трех вещественных чисел.

Программа 5:

implement main  
    open core  
domains  
    nazvanie = symbol.  
    stolica = symbol.  
    naselenie = integer.  
    territoria = real.  
  
class predicates  
    strana : (nazvanie[out], naselenie[out], territoria[out], stolica[out]) multi.  
  
clauses  
    strana("kitai", 1200, 9597000, "pekin").  
    strana("belgia", 10, 30000, "brussel").  
    strana("peru", 20, 1285000, "lima").  
  
clauses  
    run():-  
        console::init(),  
        strana(X, \_, Y, \_), Y > 1000000,  
        stdIO::writef("% - %\n", X, Y),  
        fail.  
    run().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы: 

Данная программа является примером использования переменных в запросе. Мы находим страну, территория которой больше 1000000. В результате выводим название страны и её площадь.

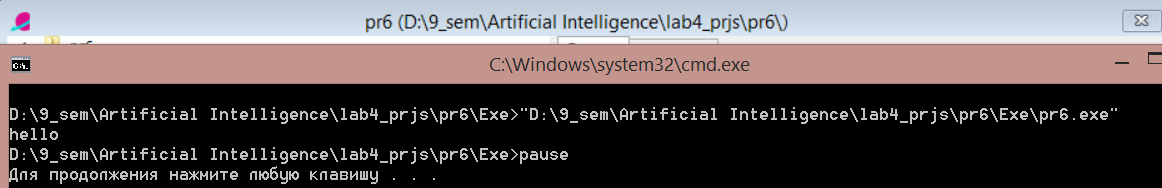
Программа 6:

implement main  
    open core  
  
clauses  
    hello():-  
        console::init(),  
        stdIO::write("hello").  
          
end implement main  
  
goal  
    main::hello().

Main.cl

class main  
    open core  
  
predicates  
    hello : ().  
  
end class main

Результаты работы:



Программа демонстрирует возможность использования своих предикатов в качестве стандартных.

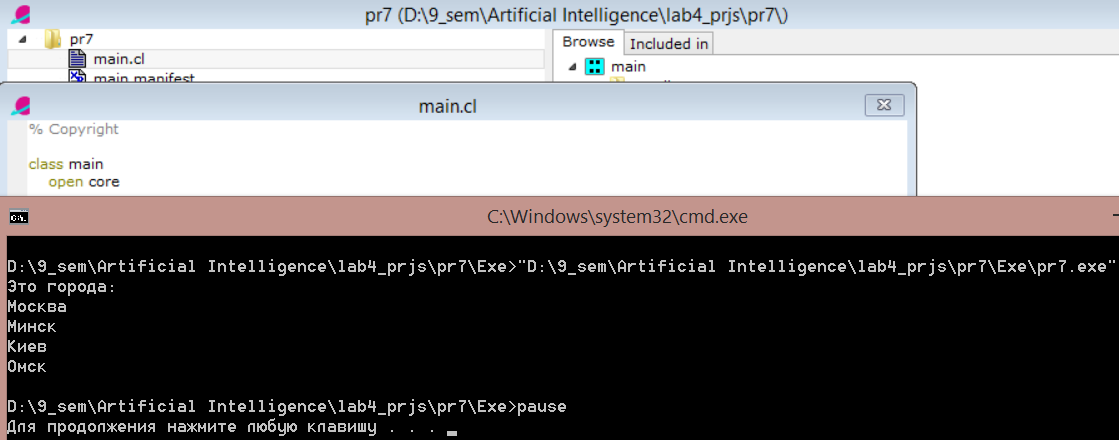
Программа 7:

implement main  
    open core  
class predicates  
    gorod : (symbol[out]) multi.  
  
clauses  
    gorod("Москва").  
    gorod("Минск").  
    gorod("Киев").  
    gorod("Омск").  
    show():-  
        gorod(X),  
        stdIO::write(X),  
        stdIO::nl(),  
        fail.  
    show().  
end implement main  
  
goal  
    console::init(),  
    stdIO::write("Это города:"),  
    stdIO::nl(),  
    main::show().

Main.cl:

class main  
    open core  
  
predicates  
    show : ().  
  
end class main

Результат работы:



Программа демонстрирует использование предиката fail, который вызывает неуспешное завершение поиска, что позволяет получить всевозможные решения задачи.

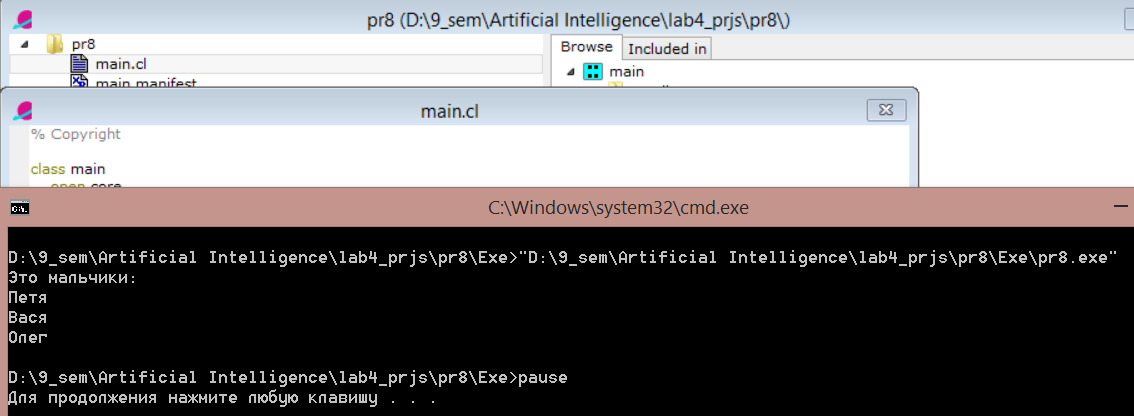
Программа 8:

implement main  
    open core      
domains  
    person = symbol.  
  
class predicates  
    deti : (person[out]) multi.  
    make\_cut : (person) determ.  
  
clauses  
    deti("Петя").  
    deti("Вася").  
    deti("Олег").  
    deti("Маша").  
    deti("Оля").  
    deti("Наташа").  
    show():-  
        deti(X),  
        stdIO::write(X),  
        stdIO::nl(),  
        make\_cut(X),  
        !.  
    show().  
    make\_cut(X):-  
        X = "Олег".  
end implement main  
  
goal  
    console::init(),  
    stdIO::write("Это мальчики:"),  
    stdIO::nl(),  
    main::show().

Main.cl

class main  
    open core  
  
predicates  
    show : ().  
  
end class main

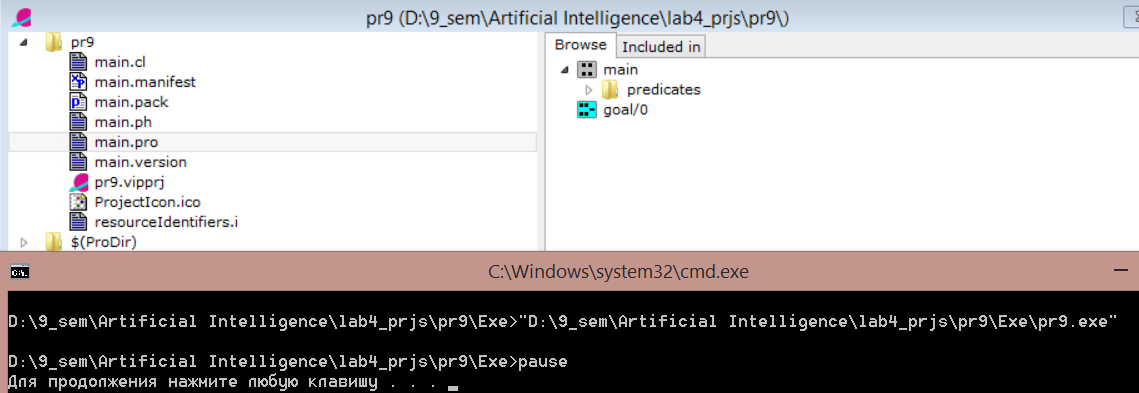
Результат работы:



Программа демонстрирует использование предиката cut,который отсекает поиск по части данных. Выводятся имена только мальчиков, так как поиск на Олеге заканчивается.

Программа 9:

implement main  
    open core  
  
class predicates  
    buy\_car : (symbol[out], symbol[out]) determ.  
    car : (symbol[out], symbol[out], integer[out]) multi.  
    color : (symbol, symbol) determ.  
  
clauses  
    car("москвич", "синий", 12000).  
    car("жигули", "зеленый", 26000).  
    car("вольво", "синий", 24000).  
    car("волга", "синий", 20000).  
    car("ауди", "зеленый", 20000).  
    color("синий", "темный").  
    color("зеленый", "светлый").  
    buy\_car(Model, Color):-  
        car(Model, Color, Price),  
        color(Color, "светлый"),  
        !,  
        Price < 25000.  
    run():-  
        console::init(),  
        buy\_car(X, Y),  
        stdIO::writef("% - %\n", X, Y),  
        fail.  
    run().  
end implement main  
  
goal  
    mainExe::run(main::run).

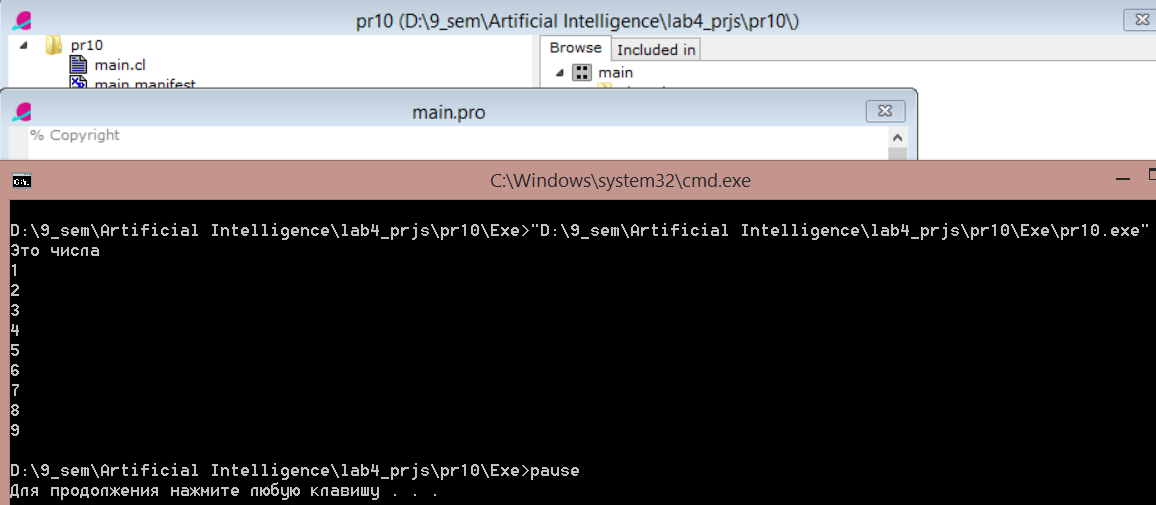
Результат работы: 

Программа не выдала решения, так как поиск заканиватеся после жигулей за 26000,а ауди стоит после жигулей в списке.

Программа 10:

implement main  
    open core  
domains  
    number = integer.  
      
class predicates  
    write\_number : (number) nondeterm.  
      
clauses  
    write\_number(10).  
    write\_number(N):-  
        N < 10,  
        stdIO::write(N),  
        stdIO::nl(),  
        write\_number(N + 1).  
    run():-  
        console::init(),  
        stdIO::write("Это числа"),  
        stdIO::nl(),  
        main::write\_number(1),  
        fail.  
    run().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

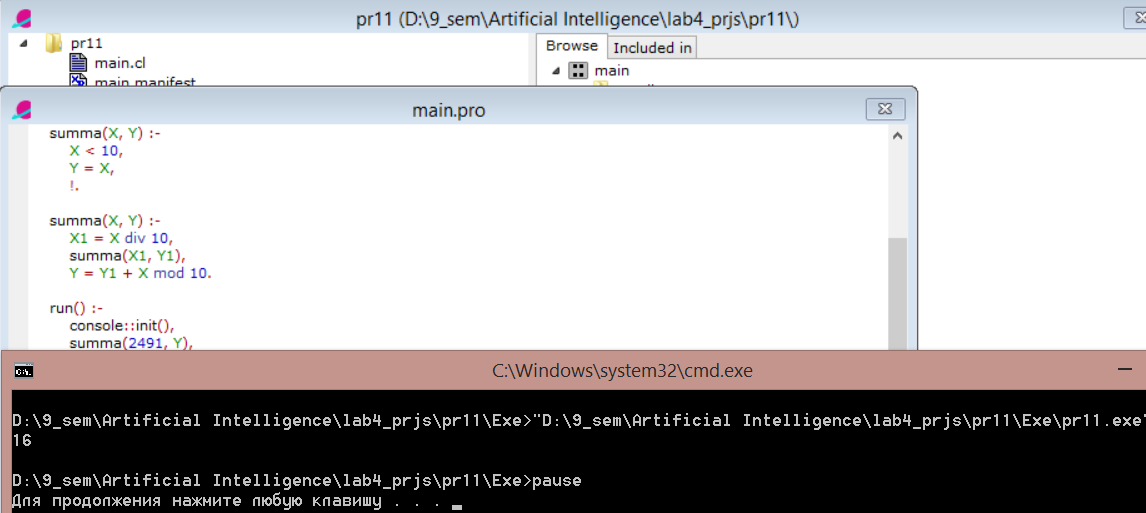


Программа демонстрирует использование рекурсии для вывода чисел с 1 по 9.

Программа 11:

implement main  
    open core  
class predicates  
    summa : (integer, integer[out]).  
clauses  
    summa(X, Y):-  
        X < 10,  
        Y = X,  
        !.  
    summa(X, Y):-  
        X1 = X div 10,  
        summa(X1, Y1),  
        Y = Y1 + X mod 10.  
    run():-  
        console::init(),  
        summa(2491, Y),  
        stdIO::write(Y),  
        stdIO::nl().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

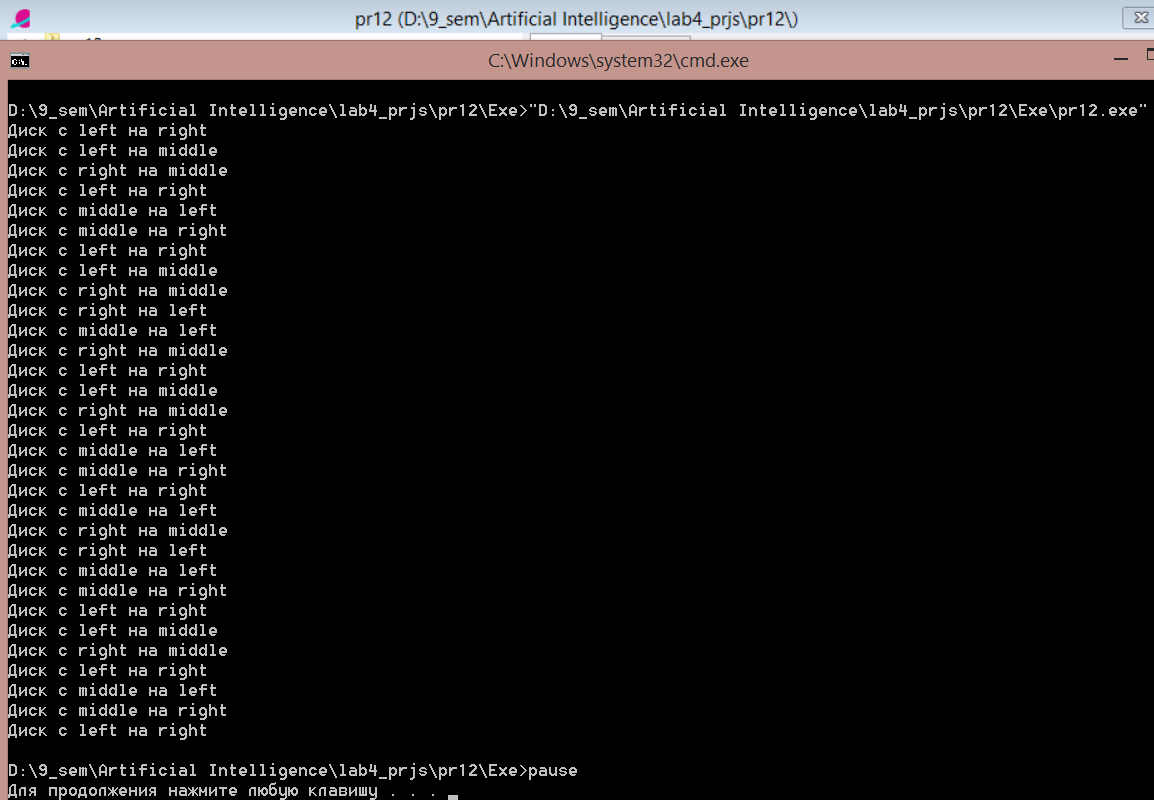


Программа использует предикат !, что позволяет избежать переполнения стека.

Программа 12:

implement main  
    open core  
  
    
domains  
    loc = right; middle; left.  
  
class predicates  
    hanoi : (integer).  
    move : (integer, loc, loc, loc).  
    inform : (loc, loc).  
  
clauses  
    hanoi(N):-  
        move(N, left, middle, right).  
    move(1, A, \_, C):-  
        inform(A, C),  
        !.  
    move(N, A, B, C):-  
        move(N - 1, A, C, B),  
        inform(A, C),  
        move(N - 1, B, A, C).  
    inform(Loc1, Loc2):-  
        stdIO::write("Диск с ", Loc1, " на ", Loc2),  
        stdIO::nl().  
    run():-  
        console::init(),  
        hanoi(5).  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

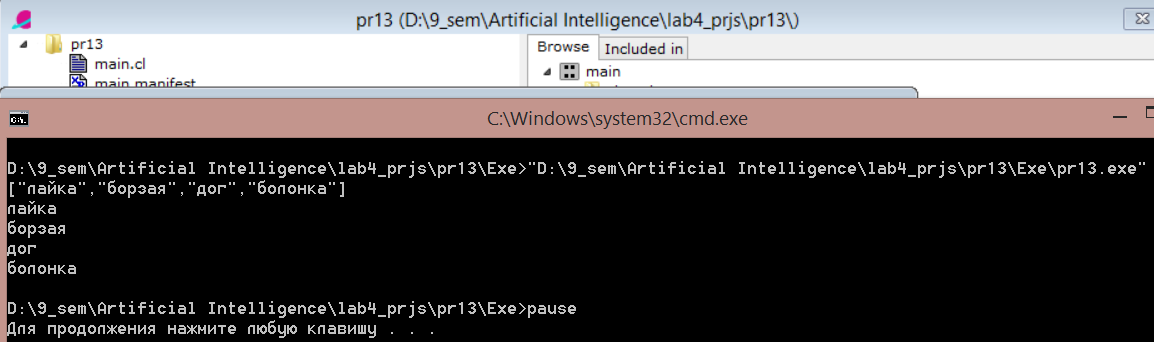


Программа является решением задачи «Ханойская башня» и выводит последовательность перемещений диска приводящая к решению.

Программа 13:

implement main  
    open core  
  
domains  
    dog\_list = symbol\*.  
      
class predicates  
    dogs : (dog\_list[out]).  
    print\_list : (dog\_list).  
  
clauses  
    dogs(["лайка", "борзая", "дог", "болонка"]).  
    print\_list([]).  
    print\_list([X|Y]):-  
        stdIO::write(X),  
        stdIO::nl(),  
        print\_list(Y).  
    run():-  
        console::init(),  
        dogs(X),  
        stdIO::write(X),  
        stdIO::nl(),  
        print\_list(X).  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

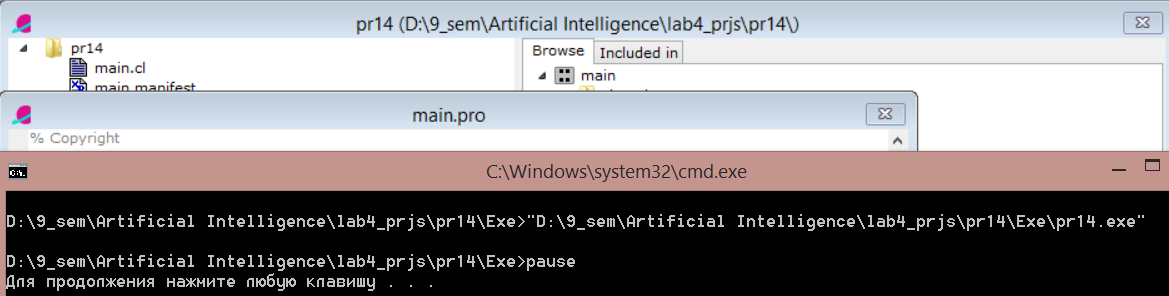


Данная прогрмма демонстрирует использование списков, для описания пород собак.

Программа 14:

implement main  
    open core  
domains  
    dog\_list = symbol\*.  
      
class predicates  
    find\_it : (symbol, dog\_list) nondeterm.  
  
clauses  
    find\_it(X, [X|\_]).  
    find\_it(X, [\_|Y]):-  
        find\_it(X, Y).  
    run():-  
        console::init(),  
        find\_it("болонка", ["лайка", "дог"]),  
        stdIO::write("да"),  
        fail.  
    run().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

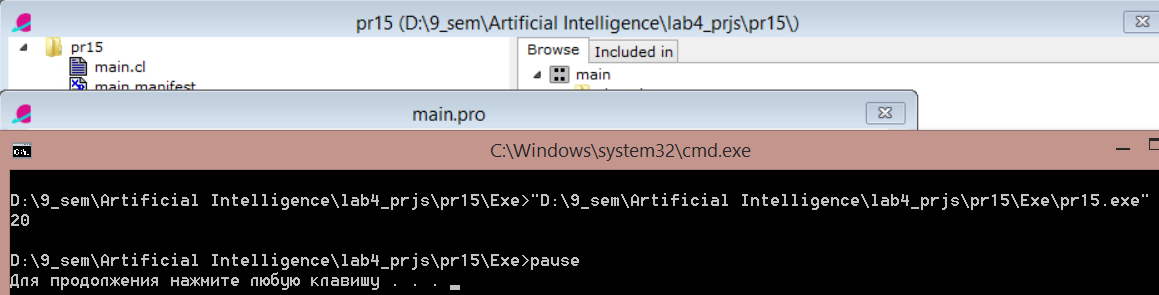


Так как болонка отсутсвует в списке собак, программа не выведет «Да».

Программа 15:

implement main  
    open core  
domains  
    spisok = integer\*.  
      
class predicates  
    summa\_sp : (spisok, integer[out]).  
  
clauses  
    summa\_sp([], 0).  
    summa\_sp([H|T], S):-  
        summa\_sp(T, S1),  
        S = H + S1.  
    run():-  
        console::init(),  
        summa\_sp([2, 5, 3, 10], Sum),  
        stdIO::write(Sum),  
        stdIO::nl().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

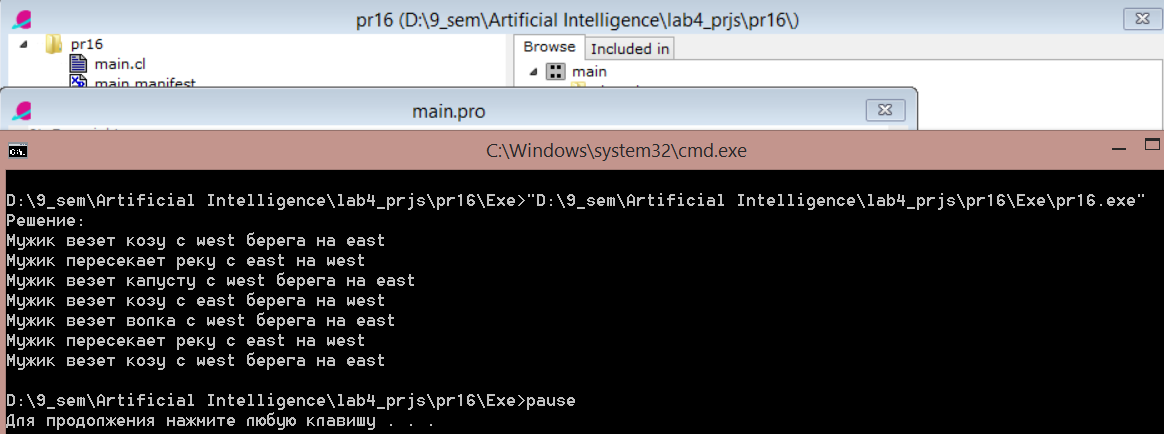


Программа подсчитывает сумму элементов в списке.

Программа 16:

implement main  
    open core  
  
domains  
    loc = east; west.  
    state = state(loc, loc, loc, loc).  
    path = state\*.  
  
class predicates  
    go : (state, state).  
    path : (state, state, path, path[out]) determ.  
    move : (state, state[out]) nondeterm.  
    opposite : (loc, loc) determ anyflow.  
    unsafe : (state) nondeterm.  
    member : (state, path) nondeterm.  
    write\_path : (path) determ.  
    write\_move : (state, state) determ.  
  
clauses  
    go(S, G):-  
        path(S, G, [S], L),  
        stdIO::write("Решение:"),  
        stdIO::nl(),  
        write\_path(L),  
        fail.  
    go(\_,\_).  
    path(S, G, L, L1):-  
        move(S, S1),  
        not(unsafe(S1)),  
        not(member(S1, L)),  
        path(S1, G, [S1|L], L1),  
        !.  
    path(G, G, T, T):-  
        !.  
    move(state(X, X, G, C), state(Y, Y, G, C)):-  
        opposite(X, Y).  
    move(state(X, W, X, C),state(Y, W, Y, C)):-  
        opposite(X, Y).  
    move(state(X, W, G, X),state(Y, W, G, Y)):-  
        opposite(X, Y).  
    move(state(X, W, G, C),state(Y, W, G, C)):-  
        opposite(X, Y).  
    opposite(east, west).  
    opposite(west, east):-  
        !.  
    unsafe(state(F, X, X, \_)):-  
        opposite(F, X).  
    unsafe(state(F, \_, X, X)):-  
        opposite(F, X).  
    member(X, [X|\_]).  
    member(X, [\_|L]):-  
        member(X, L).  
    write\_path([H1, H2|T]):-  
        !,  
        write\_move(H1, H2),  
        write\_path([H2|T]).  
    write\_path([]).  
    write\_move(state(X, W, G, C), state(Y, W, G, C)):-  
        !,  
        stdIO::write("Мужик пересекает реку с ", X, " на ", Y),  
        stdIO::nl().  
    write\_move(state(X, X, G, C), state(Y, Y, G, C)):-  
        !,  
        stdIO::write("Мужик везет волка с ", X, " берега на ", Y),  
        stdIO::nl().  
    write\_move(state(X, W, X, C), state(Y, W, Y, C)):-  
        !,  
        stdIO::write("Мужик везет козу с ", X, " берега на ", Y),  
        stdIO::nl().  
    write\_move(state(X, W, G, X), state(Y, W, G, Y)):-  
        !,  
        stdIO::write("Мужик везет капусту с ", X, " берега на ", Y),  
        stdIO::nl().  
  
  
clauses  
    run():-  
        console::init(),  
        go(state(east, east, east, east), state(west, west, west, west)).  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

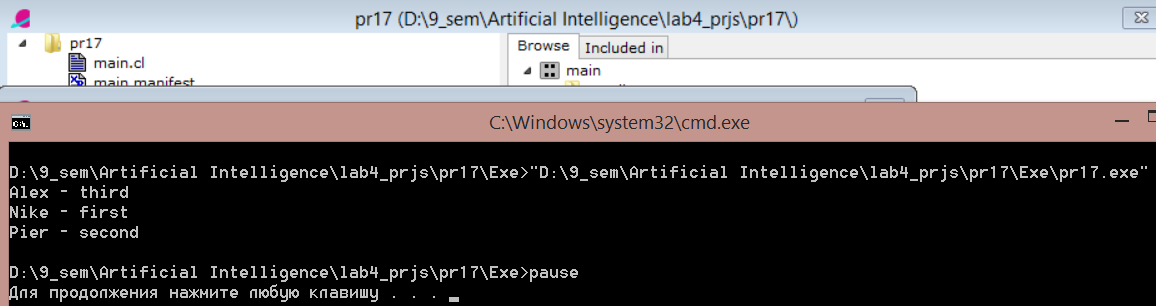


Программа демонстрирует решение задачи о волке, козе и капусте.

Программа 17:

implement main  
    open core  
class predicates  
    name : (symbol) determ.  
    name : (symbol[out]) multi.  
    mesto : (symbol) determ.  
    mesto : (symbol[out]) multi.  
    prizer : (symbol, symbol) nondeterm.  
    solution : (symbol[out], symbol[out], symbol[out], symbol[out], symbol[out], symbol[out]) determ.  
  
clauses  
    name("Alex").  
    name("Pier").  
    name("Nike").  
    mesto("first").  
    mesto("second").  
    mesto("third").  
    prizer(X, Y):-  
        name(X),  
        mesto(Y),  
        X = "Pier",  
        not (Y = "second"),  
        not (Y = "third");  
        name(X),  
        mesto(Y),  
        X = "Nike",  
        not (Y = "third");  
        name(X),  
        mesto(Y),  
        not (X = "Pier"),  
        not (X = "Nike").  
    solution(X1, Y1, X2, Y2, X3, Y3):-  
        name(X1),  
        name(X2),  
        name(X3),  
        mesto(Y1),  
        mesto(Y2),  
        mesto(Y3),  
        prizer(X1, Y1),  
        prizer(X2, Y2),  
        prizer(X2, Y3),  
        Y1 <> Y2,  
        Y2 <> Y3,  
        Y1 <> Y3,  
        X1 <> X2,  
        X2 <> X3,  
        X1 <> X3,  
        !.  
    run():-  
        console::init(),  
        solution(X1, Y1, X2, Y2, X3, Y3),  
        stdIO::writef("% - %\n% - %\n% - %\n", X1, Y1, X2, Y2, X3, Y3),  
        fail.  
    run().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

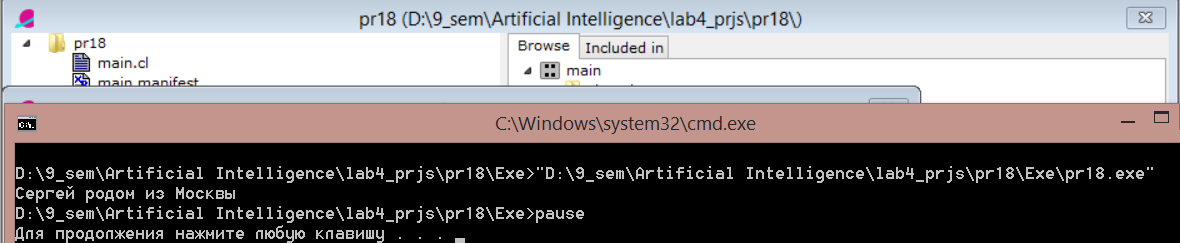


Программа решает логическую задачку по распределению призовых мест участников соревнования.

Программа 18:

implement main  
    open core  
domains  
    name = symbol.  
  
class predicates  
    student : (name) determ.  
    student : (name[out]) multi.  
    gorod : (name) determ.  
    gorod : (name[out]) multi.  
    velo : (name, name) determ.  
    fact : (name, name) determ anyflow.  
    fact1 : (name, name) determ anyflow.  
    rodom : (name, name) nondeterm.  
    rodom : (name[out], name) nondeterm.  
    rodom\_penza : (name) nondeterm.  
  
clauses  
    student(X):-  
        X = "Сергей";  
        X = "Борис";  
        X = "Виктор";  
        X = "Григорий";  
        X = "Леонид".  
    gorod(Y):-  
        Y = "Пенза";  
        Y = "Львов";  
        Y = "Москва";  
        Y = "Харьков";  
        Y = "Рига".  
    fact("Сергей", "Рига").  
    fact("Борис", "Пенза").  
    fact("Виктор", "Москва").  
    fact("Григорий", "Харьков").  
    velo(X, Y):-  
        student(X),  
        gorod(Y),  
        fact(X, Y),  
        !;  
        student(X),  
        gorod(Y),  
        not (fact(X, \_)),  
        not (fact(\_, Y)).  
    fact1("Борис", "Рига").  
    fact1("Виктор", "Львов").  
    rodom\_penza(X):-  
        student(X),  
        not (fact1(X, \_)),  
        gorod(U),  
        not (U = "Пенза"),  
        velo(X, U),  
        rodom("Леонид", U).  
    rodom(X, Z):-  
        student(X),  
        gorod(Z),  
        fact1(X, Z),  
        !;  
        student(X),  
        not (X = "Леонид"),  
        Z = "Пенза",  
        rodom\_penza(X),  
        !;  
        student(X),  
        gorod(Z),  
        not (fact1(\_, Z)),  
        X = "Леонид",  
        not (Z = "Пенза"),  
        student(K),  
        not (fact1(K, \_)),  
        velo(K, Z);  
        student(X),  
        not (X = "Леонид"),  
        gorod(Z),  
        not (Z = "Пенза"),  
        not (fact1(\_, Z)),  
        not (fact1(X, \_)),  
        gorod(Y),  
        not (Y = Z),  
        velo(X, Y),  
        not (rodom("Леонид", Z)),  
        not (rodom("Леонид", Y)).  
    run():-  
        console::init(),  
        rodom(X, "Москва"),  
        stdIO::writef("% родом из Москвы", X),  
        fail.  
    run().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:

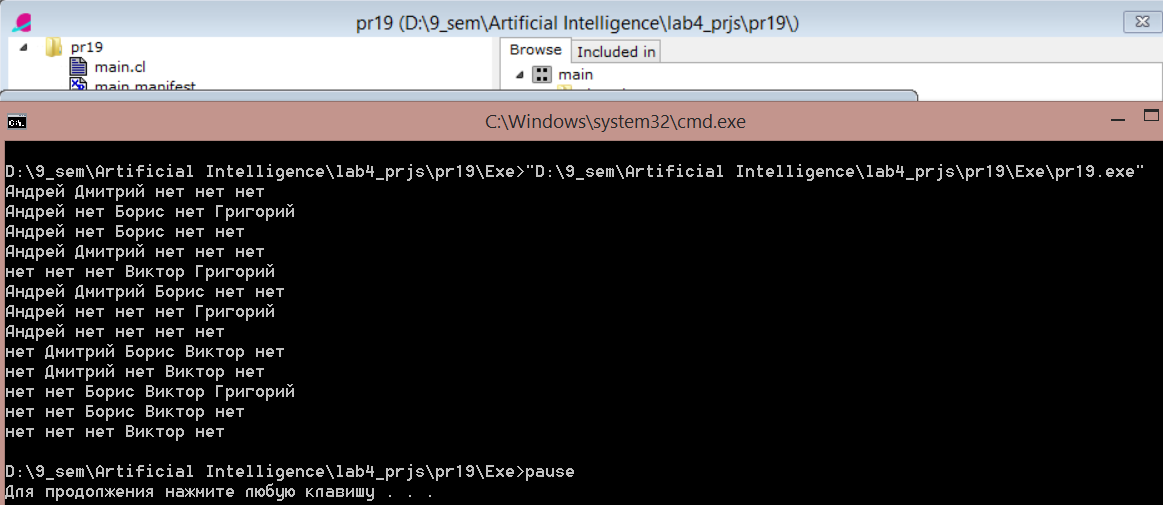


Программа демонстрирует решение задачи о месте рождения студентов.

Программа 19:

implement main  
    open core  
domains  
    s = symbol.     
class predicates  
    st\_A : (s[out]) multi.  
    st\_D : (s[out]) multi.  
    st\_B : (s[out]) multi.  
    st\_V : (s[out]) multi.  
    st\_G : (s[out]) multi.  
    ogr1 : (s, s, s, s, s) determ.  
    ogr2 : (s, s, s, s, s) determ.  
    spisok : (s[out], s[out], s[out], s[out], s[out]) nondeterm.  
    norm1 : (s, s, s, s, s) determ.  
    norm2 : (s, s, s, s, s) determ.  
    norm3 : (s, s, s, s, s) determ.  
    norm4 : (s, s, s, s, s) determ.  
clauses  
    st\_A(A):-  
        A = "Андрей";  
        A = "нет".  
    st\_D(D):-  
        D = "Дмитрий";  
        D = "нет".  
    st\_B(B):-  
        B = "Борис";  
        B = "нет".  
    st\_V(V):-  
        V = "Виктор";  
        V = "нет".  
    st\_G(G):-  
        G = "Григорий";  
        G = "нет".  
    ogr1("Андрей", \_, \_, "нет", \_).  
    ogr1("нет", \_, \_, "Виктор", \_).  
    ogr2(\_, "Дмитрий", \_, \_, "нет").  
    ogr2(\_, "нет", \_, \_, \_).  
    norm1("Андрей", "Дмитрий", "нет", \_, \_).  
    norm2("Андрей", "нет", "Борис", "нет", \_).  
    norm3(\_, "Дмитрий", "нет", "нет", \_).  
    norm4(\_, "нет", "нет", "Виктор", "Григорий").  
    spisok(A, D, B, V, G):-  
        st\_A(A),  
        st\_D(D),  
        st\_B(B),  
        st\_V(V),  
        st\_G(G),  
        norm1(A, D, B, V, G),  
        ogr1(A, D, B, V, G),  
        ogr2(A, D, B, V, G);  
        st\_A(A),  
        st\_D(D),  
        st\_B(B),  
        st\_V(V),  
        st\_G(G),  
        norm2(A, D, B, V, G),  
        ogr1(A, D, B, V, G),  
        ogr2(A, D, B, V, G);  
        st\_A(A),  
        st\_D(D),  
        st\_B(B),  
        st\_V(V),  
        st\_G(G),  
        norm3(A, D, B, V, G),  
        ogr1(A, D, B, V, G),  
        ogr2(A, D, B, V, G);  
        st\_A(A),  
        st\_D(D),  
        st\_B(B),  
        st\_V(V),  
        st\_G(G),  
        norm4(A, D, B, V, G),  
        ogr1(A, D, B, V, G),  
        ogr2(A, D, B, V, G);  
        st\_A(A),  
        st\_D(D),  
        st\_B(B),  
        st\_V(V),  
        st\_G(G),  
        not (norm1(A, D, B, V, G)),  
        not (norm2(A, D, B, V, G)),  
        not (norm3(A, D, B, V, G)),  
        not (norm4(A, D, B, V, G)),  
        ogr1(A, D, B, V, G),  
        ogr2(A, D, B, V, G).  
clauses  
    run():-  
        console::init(),  
        spisok(A, D, B, V, G),  
        stdIO::writef("% % % % %\n", A, D, B, V, G),  
        fail.  
    run().  
end implement main  
goal  
    mainExe::run(main::run).

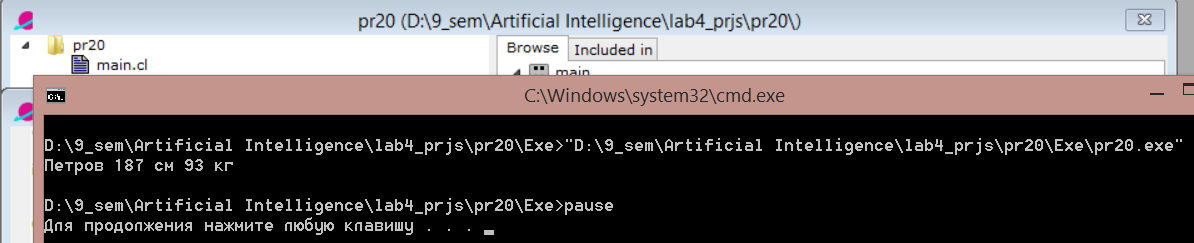
Результат работы:



Программа демонстрирует решение задачи о студентах и необходимости посещать ими лекции на каждый день недели.

Программа 20:

implement main  
    open core  
domains  
    name = symbol.  
    rost = integer.  
    ves = integer.  
  
class facts  
    dplayer : (name, rost, ves).  
  
class predicates  
    player : (name[out], rost[out], ves[out]) multi.  
    assert\_database : ().  
      
clauses  
    player("Михайлов", 180, 87).  
    player("Петров", 187, 93).  
    player("Харламов", 177, 80).  
    assert\_database():-  
        player(N, R, V),  
        assertz(dplayer(N, R, V)),  
        fail.  
    assert\_database().  
  
clauses  
    run():-  
        console::init(),  
        assert\_database(),  
        dplayer(N, R, V),  
        R > 180,  
        stdIO::writef("% % см % кг\n", N, R, V),  
        fail.  
    run().  
end implement main  
  
goal  
    mainExe::run(main::run).

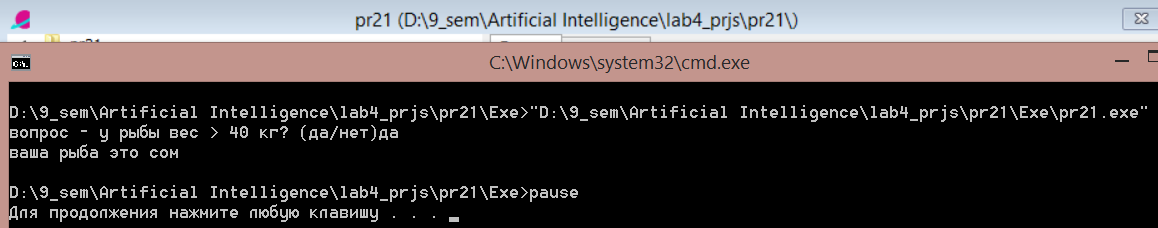
Результат работы: 

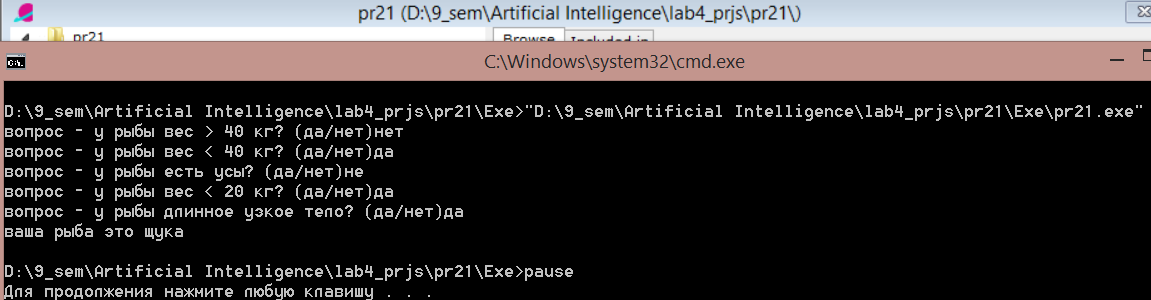
Программа демонстрирует поиск человека ростом выше 180, на роль которого подходит Петров.

Программа 21:

implement main  
    open core  
  
class facts  
    xpositive : (string, string).  
    xnegative : (string, string).  
  
class predicates  
    expertiza : ().  
    vopros : (string, string) determ.  
    fish\_is : (string[out]) nondeterm.  
    positive : (string, string) determ.  
    negative : (string, string) determ.  
    remember : (string, string, string) determ.  
    clear\_facts : ().  
  
clauses  
    expertiza():-  
        fish\_is(X),  
        !,  
        stdIO::write("ваша рыба это ", X, " "),  
        stdIO::nl,  
        clear\_facts().  
    expertiza():-  
        stdIO::write("это неизвестная рыба!"),  
        stdIO::nl,  
        clear\_facts().  
    vopros(X, Y):-  
        stdIO::write("вопрос – ", X, " ", Y, "? (да/нет)"),  
        R = stdIO::readLine(),  
        remember(X, Y, R).  
    positive(X, Y):-  
        xpositive(X, Y),  
        !.  
    positive(X, Y):-  
        not (negative(X, Y)),  
        !,  
        vopros(X, Y).  
    negative(X, Y):-  
        xnegative(X, Y),  
        !.  
    remember(X, Y, "да"):-  
        assertz(xpositive(X, Y)).  
    remember(X, Y, "нет"):-  
        assertz(xnegative(X, Y)),  
        fail.  
    clear\_facts():-  
        retract(xpositive(\_, \_)),  
        fail.  
    clear\_facts():-  
        retract(xnegative(\_, \_)),  
        fail.  
    clear\_facts().  
    fish\_is("сом"):-  
        positive("у рыбы", "вес > 40 кг").  
    fish\_is("сом"):-  
        positive("у рыбы", "вес < 40 кг"),  
        positive("у рыбы", "есть усы").  
    fish\_is("щука"):-  
        positive("у рыбы", "вес < 20 кг"),  
        positive("у рыбы", "длинное узкое тело").  
    fish\_is("окунь"):-  
        positive("у рыбы", "вес < 20 кг"),  
        positive("у рыбы", "широкое тело"),  
        positive("у рыбы", "темные полосы").  
    fish\_is("плотва"):-  
        positive("у рыбы", "вес < 20 кг"),  
        positive("у рыбы", "широкое тело"),  
        positive("у рыбы", "серебристая чешуя").  
  
clauses  
    run():-  
        console::init(),  
        expertiza().  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:





Программа реализует простую экспертную систему по определению экземпляра пойманной рыбы. На результатах была определена щука и сом, посредством продвижению по дереву решений.

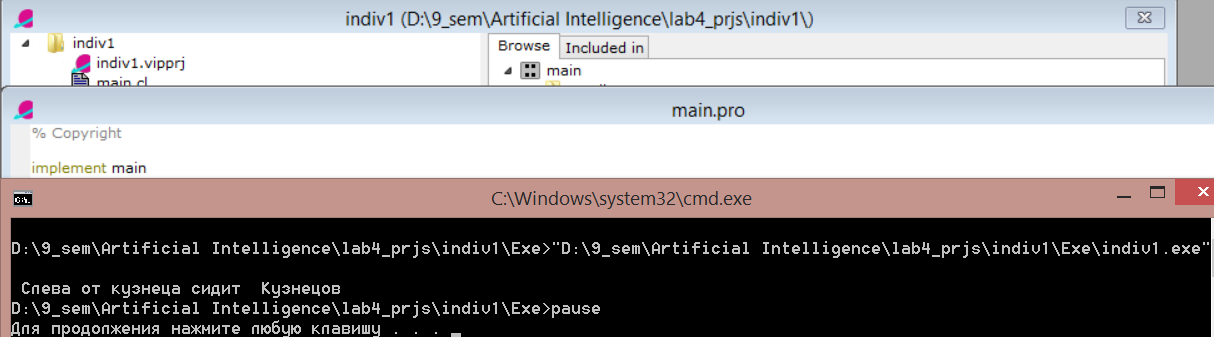
1. Выполнение индивидуального задания вар.12.[2]

Четыре человека играют в домино. Их фамилии Кузнецов, Токарев, Слесарев и Резчиков. Профессия каждого игрока соответствует фамилии одного из других игроков. Напротив Кузнецова сидит слесарь. Напротив Резчикова сидит резчик. Справа от Слесарева сидит токарь. Кто сидит слева от кузнеца?

Листинг:

% Copyright  
  
implement main  
    open core  
  
class predicates  
    solve : (symbol [out]) nondeterm.  
    name : (symbol) nondeterm.  
    name : (symbol [out]) nondeterm.  
    occupation : (symbol [out]) nondeterm.  
    occupation : (symbol) nondeterm.  
    p : (symbol, symbol) nondeterm.  
    generate : (symbol [out], symbol [out], symbol [out], symbol [out], symbol [out], symbol [out], symbol [out], symbol [out]) nondeterm.  
    check : (symbol, symbol, symbol, symbol, symbol, symbol, symbol, symbol) nondeterm.  
    left : (symbol [out], symbol, symbol, symbol, symbol, symbol, symbol, symbol, symbol, symbol) nondeterm.  
    opposite : (symbol, symbol) nondeterm.  
    left : (symbol, symbol) nondeterm.  
  
clauses  
    name("Кузнецов").  
    name("Токарев").  
    name("Слесарев").  
    name("Резчиков").  
    occupation("кузнец").  
    occupation("токарь").  
    occupation("слесарь").  
    occupation("резчик").  
    p("Кузнецов", "кузнец").  
    p("Токарев", "токарь").  
    p("Слесарев", "слесарь").  
    p("Резчиков", "резчик").  
  
    solve(X) :-  
        generate(N0, P0, N1, P1, N2, P2, N3, P3),  
        check(N0, P0, N1, P1, N2, P2, N3, P3),  
        left(X, "кузнец", N0, P0, N1, P1, N2, P2, N3, P3).  
  
    generate(N0, P0, N1, P1, N2, P2, N3, P3) :-  
        N0 = "Токарев", % здесь может быть любой    
        occupation(P0),  
        not(p(N0, P0)),  
        name(N1),  
        N1 <> N0,  
        occupation(P1),  
        P1 <> P0,  
        not(p(N1, P1)),  
        name(N2),  
        N2 <> N0,  
        N2 <> N1,  
        occupation(P2),  
        P2 <> P0,  
        P2 <> P1,  
        not(p(N2, P2)),  
        name(N3),  
        N3 <> N0,  
        N3 <> N1,  
        N3 <> N2,  
        occupation(P3),  
        P3 <> P0,  
        P3 <> P1,  
        P3 <> P2,  
        not(p(N3, P3)).  
  
    check(N0, P0, N1, P1, N2, P2, N3, P3) :-  
        opposite(N0, P2),  
        opposite(N1, P3),  
        opposite(N2, P0),  
        opposite(N3, P1),  
        left(N0, P1),  
        left(N1, P2),  
        left(N2, P3),  
        left(N3, P0).  
  
    left(N0, P1, N0, \_, \_, P1, \_, \_, \_, \_).  
  
    left(N1, P2, \_, \_, N1, \_, \_, P2, \_, \_).  
  
    left(N2, P3, \_, \_, \_, \_, N2, \_, \_, P3).  
  
    left(N3, P0, \_, P0, \_, \_, \_, \_, N3, \_).  
  
    opposite("Кузнецов", "слесарь").  
  
    opposite("Резчиков", "резчик").  
  
    opposite(N, P) :-  
        N <> "Кузнецов",  
        P <> "слесарь",  
        N <> "Резчиков",  
        P <> "резчик".  
  
    left("Слесарев", "токарь").  
  
    left(N, P) :-  
        N <> "Слесарев",  
        P <> "токарь".  
  
clauses  
    run() :-  
        console::init(),  
        solve(X),  
        stdIO::writef(" Слева от кузнеца сидит  %", X),  
        fail.  
  
    run().  
  
end implement main  
goal  
    mainExe::run(main::run).

Результат работы:



Данная программа демонстрирует решение логической задачи описанной выше. Слева от кузнеца сидит Кузнецов. Для решения необходимо было описать правила рассаживания, задать имена и профессии людей и ввести функцию проверки и начальные известные данные.

1. Изучить 1-2 лабы по методичке Седана С.Н. (доп литература)

Согласно своему варианту(3) решить задачу с помощью PROLOG**:**

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 входа, не посещая дважды одной и той же комнаты;

2.Найти путь с посещением золотой комнаты;

3.Найти путь, избегающий запрещенных к посещению комнат.

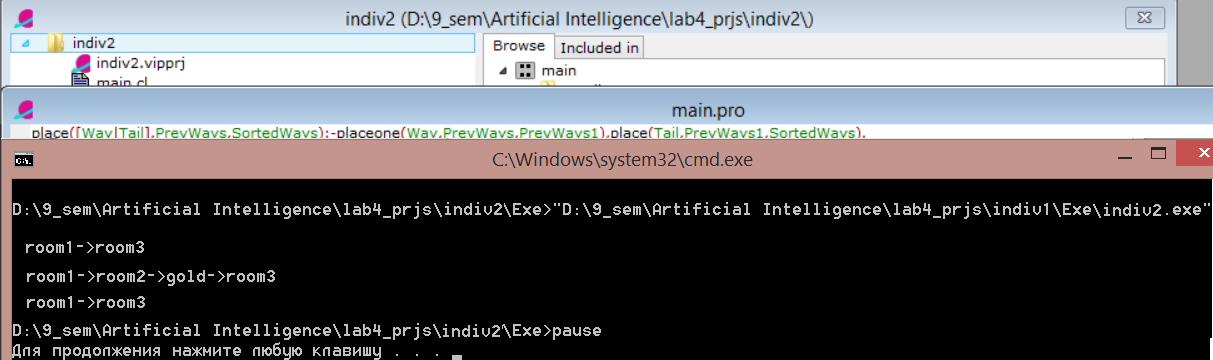
Для решения задачи было создано описание лабиринта(три комнаты,а также комната с монстрами, комната с золотом и комната с разбойниками) и реализован его обход в ширину.

Листинг:

implement main  
    open core  
  
class predicates  
    route : (symbol[out],symbol[out],symbol [out]) nondeterm.  
    route : (symbol,symbol,integer) nondeterm.  
    move : (symbo [out],symbol [out],symbol [out]) nondeterm.  
    place : (symbol [out]) nondeterm.  
    show\_answer : (symbol [out]) nondeterm.  
  
clauses  
  
route("room1","room2",4).

route("room1","room3",1).  
route("room2","gold",3).  
route("rom2","monsters",5).  
route("room2","monsters",4).  
route("gold","razb",2).  
route("razb","room3",7).  
route("monsters","room3",2).  
route("gold","room3",4).  
move("room1","room2","monsters"):-  
route("room1","room2","monsters"),route("room2","room1","monsters").  
  
  
member(H,[H|\_]).  
member(H,[\_|Tail]):-member(H,Tail).  
  
place([],SortedWays,SortedWays).  
place([Way|Tail],PrevWays,SortedWays):-placeone(Way,PrevWays,PrevWays1),place(Tail,PrevWays1,SortedWays).  
  
placeone(Length:Way,[LengthH:WayH|Tail],[Length:Way,LengthH:WayH|Tail]):-Length=<LengthH,!.  
placeone(LengthWay,[LengthHWayH|Tail],[LengthHWayH|NewTail]):-placeone(LengthWay,Tail,NewTail).  
placeone(LengthWay,[],[LengthWay]).  
  
search\_bst(Start,Finish):-  
bst([0:[Start]],Finish,Length:Way),  
show\_answer(Way),nl,write( ‘-> ') , show\_answer(Tail).  
  
bst([Length:[Finish|Tail]|\_],Finish,Length:[Finish|Tail]).  
bst([TempWay|OtherWays],Finish,Way):-  
findall(W,prolong(TempWay,W),Ways),  
place(Ways,OtherWays,NewWays),  
bst(NewWays,Finish,Way).  
  
clauses  
    run() :-  
        console::init(),  
  
show\_answer([\_]).  
show\_answer("gold").  
show\_answer(["monsters","razb"]),   
fail.  
  
    run().  
  
end implement main  
  
goal  
    mainExe::run(main::run).

Результат работы:



В результате работы возвращаются наиболее простые пути следования, так как лабиринт задан простой- пути прохода также простые.

Вывод:

В ходе лабораторной работы я изучил основы нового для себя логического языка Prolog. Данный язык оказался весьма удобен для формального описания различных логических задач и нахождения их решения.

**Ответы на вопросы:**

**1. В чем Плюсы и минусы языка Prolog?**

Данный язык является логическим, и как следствие, направлен на решение логических задач(задачи по матлогике, доказательства теорем, символьных вычислений и пр.). Он плохо подходит для вычисления математических выражений и функций, однако для этого есть специальные средства. Кроме того, неоспоримым плюсом является развитие данного языка(хотя и не такое значительное, как у популярных языков программирования), имеется большое количество интернет ресурсов с мануалами, а актуальная версия среды поддерживается современными ос, что повышает комфорт использования.

**2. Какие еще языки используются для разработки ИИ, приведите примеры (НЕ МЕ-**

**НЕЕ 2-х) проектов, языков и краткое описание проектов. (Альтернативы PROLOG)**

|  |  |  |
| --- | --- | --- |
| **Проект** | **Язык** | **Описание** |
| ODASE(http://www.odaseontologies.com/) | Mercury | Проект, развиваемый компанией по созданию программного обеспечения «Mission Critical IT». Язык используется для исследовательской платформы по созданию онтологий. |
| AutoCAD | LISP(AutoLISP) | Язык используется для написания скриптов в AutoCAD. |

**3. Решаема ли проблема комбинаторного взрыва, пути решения?**

Комбинаторный взрыв- резкий рост временной сложности алгоритма при увеличении числа входных данных задачи. Данная проблема до сих пор н решена, но в каждом частном случае возможны варианты использования специфического для данной задачи эвристического алгоритма.

**4. Корректно ли по-вашему в принципе разработка языка ИИ? Что он должен из себя**

**представлять?**

Да, корректна, так как продвижение в данной области накапливает знания об удобстве тех или иных конструкций. С развитием ИИ должен развиваться и язык. Он должен представлять из себя переосмысленный набор уже реализованных конструкций(на предмет необходимости использования).

**5. Можно ли разработать ИИ не понимая, как он работает, должны ли мы понимать,**

**как он работает, думает, рассуждает?**

Можно, по аналогии с описанием правил нахождения правильного решения, но не самого решения какой-либо задачи. Однако, такая разработка влечёт за собой невозможность верификации, и как следствие – усложнение внешних систем контроля над конкретным продуктом, так как достоверно нельзя будет утверждать что он работает корректно. Тем не менее, человечество уже использует вещи, которые не подвергаются верификации (все ячейки ПЗУ невозможно проверить на корректную работу во всех вариациях данных за удовлетворительное время), и придуманы средства проверки таких продуктов, а значит такое решение возможно и для ИИ.

Список используемой литературы

[1] Основы системы Visual Prolog [Электронный ресурс]. –– URL: http://wikiru.visual-prolog.com/index.php?title=%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B\_%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B\_Visual\_Prologpdf (дата обращения:30.10.2017).

[2] Бураков С. В. «Язык логического программирования PROLOG», СПбГУАП,2003, [Электронный ресурс]. –– URL:http://www.ict.edu.ru/ft/005578/byrakov.pdf(дата обращения:30.10.2017).