**Цель работы:** овладеть навыками трассировки логических программ на примере приложения PIE (Prolog Inference Engine - Машина Вывода Пролога).

**Порядок выполнения работы** каждой из приведенных ниже задач следующий:

1. Загрузить программу в окно редактирования.

2. Задать необходимые исходные данные.

3. Включить режим трассировки.

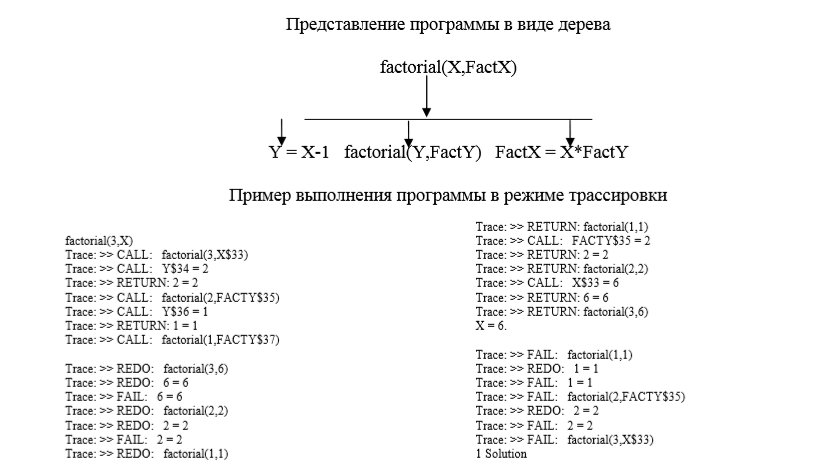
4. Сформировать цель и выполнить программу.

5. Проанализировать полученный результат.

6. Оформить отчет по работе.

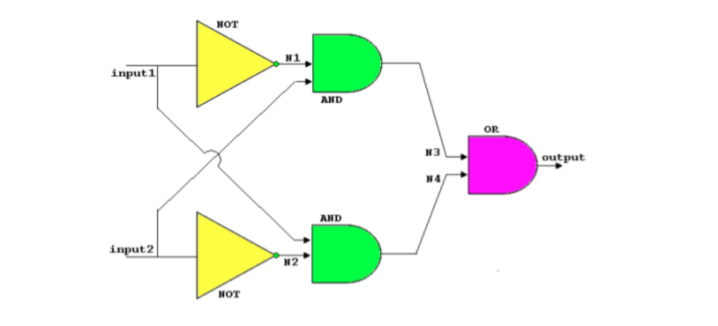
Задача 1. Программа вычисления факториала:

% predicates   
% factorial(integer, real)  
% clauses  
  
factorial(1, 1) :- !.  
factorial(X, FactX) :- Y = X-1,   
                                factorial(Y, FactY),    
                                FactX = X\*FactY.  
goal:- factorial(5, FactX), write(FactX), nl.



Задача 2. Моделирование элементов аппаратуры

Любая логическая цепь может быть представлена в Visual Prolog при помощи предикатов, где они описывают соотношения между входными и выходными сигналами. Основные элементы логики могут быть описаны с помощью не только внешних, но и внутренних связей. В качестве примера сконструируем элемент «исключающее или» из элементов AND, OR и NOT, а затем проверим его работу с помощью полученной программы. Проинтерпретировав результат как таблицу истинности, вы увидите, что указанный элемент действительно выполняет требуемую задачу.



not\_(1,0).   
not\_(0,1).  
and\_(0,0,0).  
and\_(0,1,0).  
and\_(1,0,0).   
and\_(1,1,1).  
or\_(0,0,0).  
or\_(0,1,1).  
or\_(1,0,1).  
or\_(1,1,1).  
  
xor(Input1,Input2,Output):- not\_(Input1,N1),   
                                            not\_(Input2,N2),   
                                            and\_(Input1,N2,N3),   
                                            and\_(Input2,N1,N4),   
                                            or\_(N3,N4,Output).

Представление программы в виде дерева:

Выполнение программы в режиме трассировки:

xor(1, 1, Output)  
Trace: >> CALL: xor(1,1,OUTPUT$18)  
Trace: >> CALL: not\_(1,N1$19)  
Trace: >> RETURN: not\_(1,0)  
Trace: >> CALL: not\_(1,N2$20)  
Trace: >> RETURN: not\_(1,0)  
Trace: >> CALL: and\_(1,0,N3$21)  
Trace: >> RETURN: and\_(1,0,0)  
Trace: >> CALL: and\_(1,0,N4$22)  
Trace: >> RETURN: and\_(1,0,0)  
Trace: >> CALL: or\_(0,0,OUTPUT$18)  
Trace: >> RETURN: or\_(0,0,0)  
Trace: >> RETURN: xor(1,1,0)  
OUTPUT = 0.  
  
Trace: >> REDO: xor(1,1,0)  
Trace: >> REDO: or\_(0,0,0)  
Trace: >> FAIL: or\_(0,0,OUTPUT$18)  
Trace: >> REDO: and\_(1,0,0)  
Trace: >> FAIL: and\_(1,0,N4$22)  
Trace: >> REDO: and\_(1,0,0)  
Trace: >> FAIL: and\_(1,0,N3$21)  
Trace: >> REDO: not\_(1,0)  
Trace: >> FAIL: not\_(1,N2$20)  
Trace: >> REDO: not\_(1,0)  
Trace: >> FAIL: not\_(1,N1$19)  
Trace: >> FAIL: xor(1,1,OUTPUT$18)  
1 Solution

Задача 3. Ханойские башни

Решение задачи "Ханойские башни" – классический пример рекурсии. Эта древняя игра состоит из набора деревянных дисков и трех стержней, прикрепленных к основанию. Каждый диск имеет свой диаметр, а в середине отверстие, достаточное, чтобы диск можно было надеть на стержень. В начале игры все диски надеты на левый стержень в том порядке, как это показано на рисунке. Цель игры заключается в том, чтобы перенести все диски с левого на правый стержень, по одному диску за раз, при этом ни один больший диск нельзя ставить сверху на меньший по диаметру.

Эту задачу можно легко выполнить для одного или двух дисков, но, когда число дисков становится больше трех, задача усложняется. Используем язык Пролог для ее решения. Для этой игры есть простая стратегия:

- один диск перемещается непосредственно;

- N дисков переносятся в три этапа:

* перенести N-1 дисков на средний стержень;
* перенести последний диск на первый стержень;
* перенести N-1 дисков со среднего стержня на правый.

В нашей программе есть три предиката:

- предикат **hanoi** с одним параметром, указывающим, со сколькими дисками мы играем;

- предикат **move**, с помощью которого мы описываем перенос N дисков с одного стержня на другой, используя третий стержень в качестве промежуточного места для дисков;

- предикат **inform**, который указывает на действие, выполняемое с конкретным диском. Проанализируем программу.

hanoi(N):- move(N,left,middle,right).  
move(1,A,\_,C):- inform(A,C), !.  
move(N,A,B,C):- N1=N-1,   
                            move(N1,A,C,B),   
                            inform(A,C),   
                            move(N1,B,A,C).  
inform(Loc1,Loc2):- write("Move a disk from ", Loc1," to ",Loc2), nl.

Представление программы в виде дерева:

Выполнение программы в режиме трассировки:

hanoi(3)

Trace: >> CALL: hanoi(3)  
Trace: >> CALL: move(3,left,middle,right)  
Trace: >> CALL: N1$12 = 2  
Trace: >> RETURN: 2 = 2  
Trace: >> CALL: move(2,left,right,middle)  
Trace: >> CALL: N1$13 = 1  
Trace: >> RETURN: 1 = 1  
Trace: >> CALL: move(1,left,middle,right)  
Trace: >> CALL: inform(left,right)  
Trace: >> CALL: write(Move a disk from ,left, to ,right)  
Move a disk from left to rightTrace: >> RETURN: write(Move a disk from ,left, to ,right)  
Trace: >> CALL: nl()  
  
Trace: >> RETURN: nl()  
Trace: >> RETURN: inform(left,right)  
Trace: >> RETURN: move(1,left,middle,right)  
Trace: >> CALL: inform(left,middle)  
Trace: >> CALL: write(Move a disk from ,left, to ,middle)  
Move a disk from left to middleTrace: >> RETURN: write(Move a disk from ,left, to ,middle)  
Trace: >> CALL: nl()  
  
Trace: >> RETURN: nl()  
Trace: >> RETURN: inform(left,middle)  
Trace: >> CALL: move(1,right,left,middle)  
Trace: >> CALL: inform(right,middle)  
Trace: >> CALL: write(Move a disk from ,right, to ,middle)  
Move a disk from right to middleTrace: >> RETURN: write(Move a disk from ,right, to ,middle)  
Trace: >> CALL: nl()  
  
Trace: >> RETURN: nl()  
Trace: >> RETURN: inform(right,middle)  
Trace: >> RETURN: move(1,right,left,middle)  
Trace: >> RETURN: move(2,left,right,middle)  
Trace: >> CALL: inform(left,right)  
Trace: >> CALL: write(Move a disk from ,left, to ,right)  
Move a disk from left to rightTrace: >> RETURN: write(Move a disk from ,left, to ,right)  
Trace: >> CALL: nl()  
  
Trace: >> RETURN: nl()  
Trace: >> RETURN: inform(left,right)  
Trace: >> CALL: move(2,middle,left,right)  
Trace: >> CALL: N1$14 = 1  
Trace: >> RETURN: 1 = 1  
Trace: >> CALL: move(1,middle,right,left)  
Trace: >> CALL: inform(middle,left)  
Trace: >> CALL: write(Move a disk from ,middle, to ,left)  
Move a disk from middle to leftTrace: >> RETURN: write(Move a disk from ,middle, to ,left)  
Trace: >> CALL: nl()  
  
Trace: >> RETURN: nl()  
Trace: >> RETURN: inform(middle,left)  
Trace: >> RETURN: move(1,middle,right,left)  
Trace: >> CALL: inform(middle,right)  
Trace: >> CALL: write(Move a disk from ,middle, to ,right)  
Move a disk from middle to rightTrace: >> RETURN: write(Move a disk from ,middle, to ,right)  
Trace: >> CALL: nl()  
  
Trace: >> RETURN: nl()  
Trace: >> RETURN: inform(middle,right)  
Trace: >> CALL: move(1,left,middle,right)  
Trace: >> CALL: inform(left,right)  
Trace: >> CALL: write(Move a disk from ,left, to ,right)  
Move a disk from left to rightTrace: >> RETURN: write(Move a disk from ,left, to ,right)  
Trace: >> CALL: nl()  
  
Trace: >> RETURN: nl()  
Trace: >> RETURN: inform(left,right)  
Trace: >> RETURN: move(1,left,middle,right)  
Trace: >> RETURN: move(2,middle,left,right)  
Trace: >> RETURN: move(3,left,middle,right)  
Trace: >> RETURN: hanoi(3)

True  
1 Solution