Завдання 2.1, Якщо карта має регіони N, то оцінюють, скільки обчислень, можливо, доведеться зробити для того, щоб визначити чи або не забарвлення знаходиться в конфлікті. Аргументуйте використання дерев програмної пропозиції.

(Упражнение 2.1. Если карта имеет N областей, оцените, сколько вычислений может потребоваться сделать, чтобы определить, конфликтует ли раскраска. Аргументируйте, используя деревья предложений программы.)

Тут найбільше меж коли регіони - сектори круга, що мають спільну точку. Кількість конфліктів, що треба перевірити = кількості меж, їх N(N-1)/2. Враховуючи, що в дереві перевірки конфлікту перевіряється по три предикати. То загальна кількість операцій(перевірок предикатів) не більше 3*N(N-1)/2.

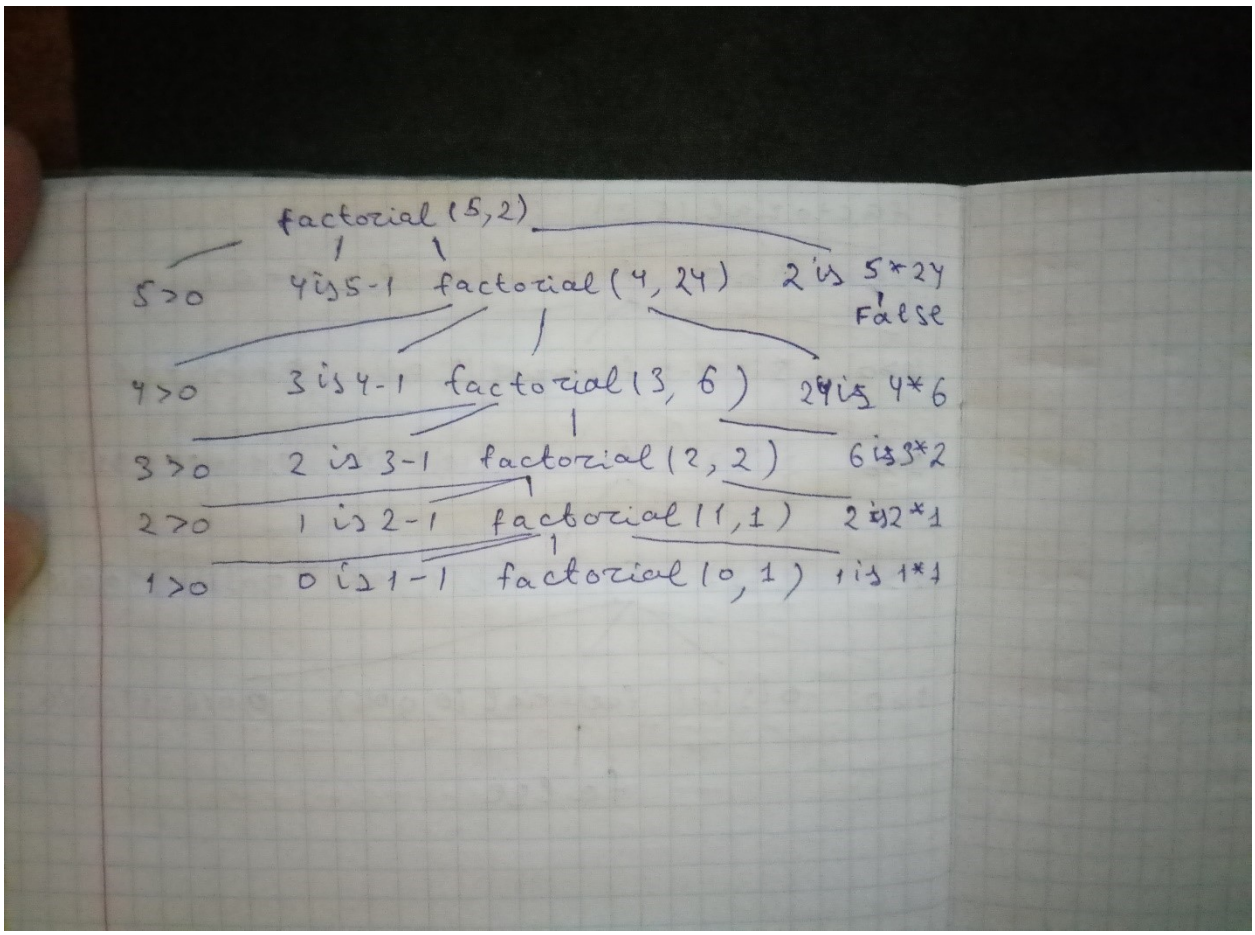
Вправа 2.2.1, Використовуючи першу фабричну програму, показують явно, що не може можливо бути дерева пропозиції, упровадженого в ''factorial(5,2) '', що має всі дійсні листи.

(*Упражнение 2.2.1.* Используя первую факториальную программу, явно покажите, что не может быть дерева предложений с корнем 'factorial(5,2)', имеющего все истинные листья.)

Подивимось на обхід дерева і побудуємо по ньому дерево:

```
[trace]  ?- factorial(5,2).
   Call: (10) factorial(5, 2) ? creep
   Call: (11) 5>0 ? creep
   Exit: (11) 5>0 ? creep
   Call: (11) _5314 is 5+ -1 ? creep
   Exit: (11) 4 is 5+ -1 ? creep
   Call: (11) factorial(4, _6824) ? creep
   Call: (12) 4>0 ? creep
   Exit: (12) 4>0 ? creep
   Call: (12) _9094 is 4+ -1 ? creep
   Exit: (12) 3 is 4+ -1 ? creep
   Call: (12) factorial(3, _10604) ? creep
   Call: (13) 3>0 ? creep
   Exit: (13) 3>0 ? creep
   Call: (13) _12874 is 3+ -1 ? creep
   Exit: (13) 2 is 3+ -1 ? creep
   Call: (13) factorial(2, _14384) ? creep
   Call: (14) 2>0 ? creep
   Exit: (14) 2>0 ? creep
   Call: (14) _16654 is 2+ -1 ? creep
   Exit: (14) 1 is 2+ -1 ? creep
   Call: (14) factorial(1, _18164) ? creep
   Call: (15) 1>0 ? creep
   Exit: (15) 1>0 ? creep
   Call: (15) _20434 is 1+ -1 ? creep
   Exit: (15) 0 is 1+ -1 ? creep
   Call: (15) factorial(0, _21944) ? creep
   Exit: (15) factorial(0, 1) ? creep
   Call: (15) _18164 is 1*1 ? creep
   Exit: (15) 1 is 1*1 ? creep
   Exit: (14) factorial(1, 1) ? creep
   Call: (14) _14384 is 2*1 ? creep
   Exit: (14) 2 is 2*1 ? creep
   Exit: (13) factorial(2, 2) ? creep
   Call: (13) _10604 is 3*2 ? creep
   Exit: (13) 6 is 3*2 ? creep
   Exit: (12) factorial(3, 6) ? creep
   Call: (12) _6824 is 4*6 ? creep
   Exit: (12) 24 is 4*6 ? creep
   Exit: (11) factorial(4, 24) ? creep
   Call: (11) 2 is 5*24 ? creep
   Fail: (11) 2 is 5*24 ?
```

По результатам трасіровки побудував дерево і показав в якому вузлі був результат fail (false).



Вправа 2.2.2 Зволікають дерево пропозиції для goal ''factorial(3,1,6) '', що має всі дійсні листи, за модою, подібною до цього, зробленому для факторіалу(3,6) заздалегідь. Як дві програми відрізняються відносно того, як вони обчислюють факторіал? Також, стежте за goal ''factorial(3,1,6) '', використовуючи Пролог.

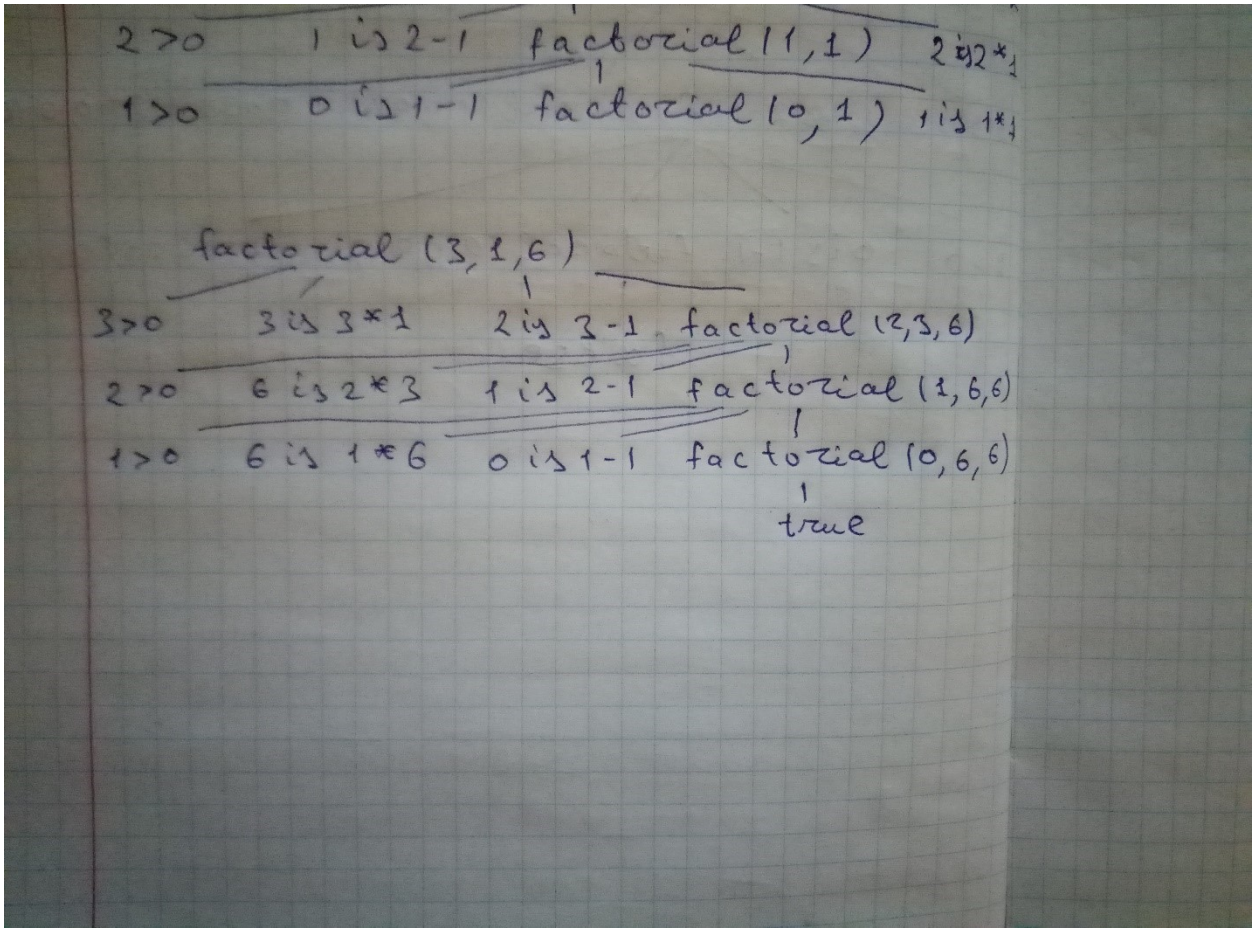
(*Упражнение 2.2.2* Нарисуйте дерево предложений для цели 'factorial(3,1,6)', имеющее все истинные листья, аналогично тому, как это делалось ранее для factorial(3,6). Чем эти две программы отличаются в том, как они вычисляют факториал? Кроме того, проследите цель «факториал (3,1,6)» с помощью Пролога.)

Простежив трасіровкой за виконанням

? - factorial(3,1,6).

```
[trace]  ?- factorial(3,1,6).
   Call: (10) factorial(3, 1, 6) ? creep
   Call: (11) 3>0 ? creep
   Exit: (11) 3>0 ? creep
   Call: (11) _3368 is 3*1 ? creep
   Exit: (11) 3 is 3*1 ? creep
   Call: (11) _4890 is 3+ -1 ? creep
   Exit: (11) 2 is 3+ -1 ? creep
   Call: (11) factorial(2, 3, 6) ? creep
   Call: (12) 2>0 ? creep
   Exit: (12) 2>0 ? creep
   Call: (12) _8666 is 2*3 ? creep
   Exit: (12) 6 is 2*3 ? creep
   Call: (12) _10188 is 2+ -1 ? creep
   Exit: (12) 1 is 2+ -1 ? creep
   Call: (12) factorial(1, 6, 6) ? creep
   Call: (13) 1>0 ? creep
   Exit: (13) 1>0 ? creep
   Call: (13) _13964 is 1*6 ? creep
   Exit: (13) 6 is 1*6 ? creep
   Call: (13) _15486 is 1+ -1 ? creep
   Exit: (13) 0 is 1+ -1 ? creep
   Call: (13) factorial(0, 6, 6) ? creep
   Exit: (13) factorial(0, 6, 6) ? creep
   Exit: (12) factorial(1, 6, 6) ? creep
   Exit: (11) factorial(2, 3, 6) ? creep
   Exit: (10) factorial(3, 1, 6) ? creep
true .
```

Побудував дерево(по трасіровці):



Побудувавши дерево по другому факторіалу, що застосовує акомулятор і хвостову рекурсію я побачив, що він дає результат, по проходженню вниз по дереву від кореня до листа типу factorial(0,F,F) (далі він піднімався, але математичних операцій не виконував, лише логічне присвоєння хибності

факторіалам виконував аж до кореня як коньнкюції синів), а перший факторіал, після опускання до нижнього листа, починав підніматися і продовжувати розрахунки, і давав результат лише після повернення до самого правого нащадку кореня (останній математичний розрахунок там був), далі лише присвоєння результату істинності кореню як кон’юнкції синів кореня(перших нащадків).

Вправа 2.3.1 Зволікають дерево програмної пропозиції для goal ''move(3,left,right,center) '' показують, що це - наслідок програми. Як дерево цієї пропозиції пов'язане з підстановкою обробляють пояснюється вище?

(*Упражнение 2.3.1* Нарисуйте дерево предложений программы для цели «переместить (3, влево, вправо, в центр)» и покажите, что это следствие программы. Как это дерево предложений связано с описанным выше процессом замены?)

Ось результат виконання:

```
?- move(3,left,right,center).
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right
true
```

Провів трасіровку:

```
true.

[trace]  ?- move(3,left,right,center).
   Call: (10) move(3, left, right, center) ? creep
   Call: (11) 3>1 ? creep
   Exit: (11) 3>1 ? creep
   Call: (11) _10406 is 3+ -1 ? creep
   Exit: (11) 2 is 3+ -1 ? creep
   Call: (11) move(2, left, center, right) ? creep
   Call: (12) 2>1 ? creep
   Exit: (12) 2>1 ? creep
   Call: (12) _14192 is 2+ -1 ? creep
   Exit: (12) 1 is 2+ -1 ? creep
   Call: (12) move(1, left, right, center) ? creep
   Call: (13) write('Move top disk from ') ? creep
Move top disk from
   Exit: (13) write('Move top disk from ') ? creep
   Call: (13) write(left) ? creep
left
   Exit: (13) write(left) ? creep
   Call: (13) write(' to ') ? creep
 to
   Exit: (13) write(' to ') ? creep
   Call: (13) write(right) ? creep
right
   Exit: (13) write(right) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) move(1, left, right, center) ? creep
   Call: (12) move(1, left, center, _24772) ? creep
   Call: (13) write('Move top disk from ') ? creep
Move top disk from
   Exit: (13) write('Move top disk from ') ? creep
   Call: (13) write(left) ? creep
left
   Exit: (13) write(left) ? creep
   Call: (13) write(' to ') ? creep
 to
   Exit: (13) write(' to ') ? creep
   Call: (13) write(center) ? creep
center
   Exit: (13) write(center) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) move(1, left, center, _1576) ? creep
   Call: (12) move(1, right, center, left) ? creep
   Call: (13) write('Move top disk from ') ? creep
Move top disk from
   Exit: (13) write('Move top disk from ') ? creep
   Call: (13) write(right) ? creep
right
   Exit: (13) write(right) ? creep
   Call: (13) write(' to ') ? creep
 to
   Exit: (13) write(' to ') ? creep
   Call: (13) write(center) ? creep
center
   Exit: (13) write(center) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) move(1, right, center, left) ? creep
   Exit: (11) move(2, left, center, right) ? creep
   Call: (11) move(1, left, right, _12108) ? creep
   Call: (12) write('Move top disk from ') ? creep
Move top disk from
   Exit: (12) write('Move top disk from ') ? creep
   Call: (12) write(left) ? creep
```

```
left
   Exit: (12) write(left) ? creep
   Call: (12) write(' to ') ? creep
 to
   Exit: (12) write(' to ') ? creep
   Call: (12) write(right) ? creep
right
   Exit: (12) write(right) ? creep
   Call: (12) nl ? creep

   Exit: (12) nl ? creep
   Exit: (11) move(1, left, right, _20354) ? creep
   Call: (11) move(2, center, right, left) ? creep
   Call: (12) 2>1 ? creep
   Exit: (12) 2>1 ? creep
   Call: (12) _23330 is 2+ -1 ? creep
   Exit: (12) 1 is 2+ -1 ? creep
   Call: (12) move(1, center, left, right) ? creep
   Call: (13) write('Move top disk from ') ? creep
Move top disk from
   Exit: (13) write('Move top disk from ') ? creep
   Call: (13) write(center) ? creep
center
   Exit: (13) write(center) ? creep
   Call: (13) write(' to ') ? creep
 to
   Exit: (13) write(' to ') ? creep
   Call: (13) write(left) ? creep
left
   Exit: (13) write(left) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) move(1, center, left, right) ? creep
   Call: (12) move(1, center, right, _2374) ? creep
   Call: (13) write('Move top disk from ') ? creep
Move top disk from
   Exit: (13) write('Move top disk from ') ? creep
   Call: (13) write(center) ? creep
center
   Exit: (13) write(center) ? creep
   Call: (13) write(' to ') ? creep
 to
   Exit: (13) write(' to ') ? creep
   Call: (13) write(right) ? creep
right
   Exit: (13) write(right) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) move(1, center, right, _10620) ? creep
   Call: (12) move(1, left, right, center) ? creep
   Call: (13) write('Move top disk from ') ? creep
Move top disk from
   Exit: (13) write('Move top disk from ') ? creep
   Call: (13) write(left) ? creep
left
   Exit: (13) write(left) ? creep
   Call: (13) write(' to ') ? creep
 to
   Exit: (13) write(' to ') ? creep
   Call: (13) write(right) ? creep
right
   Exit: (13) write(right) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) move(1, left, right, center) ? creep
   Exit: (11) move(2, center, right, left) ? creep
   Exit: (10) move(3, left, right, center) ? creep
true .

[trace]  ?-
```

Піпец яка довга. І по ній мені дерево побудувати? Я попробував будувати, але воно виявилось широким і не влізло на листок. І на 2 листка не влізло. І навіть якщо не розписувати одиничне

переміщення, також на два листкане влазе. А якщо скоротити назви до 1 букви, то влазе на два листка, тому я побудува дерево, використовуючи наступні скорочення:

move – m

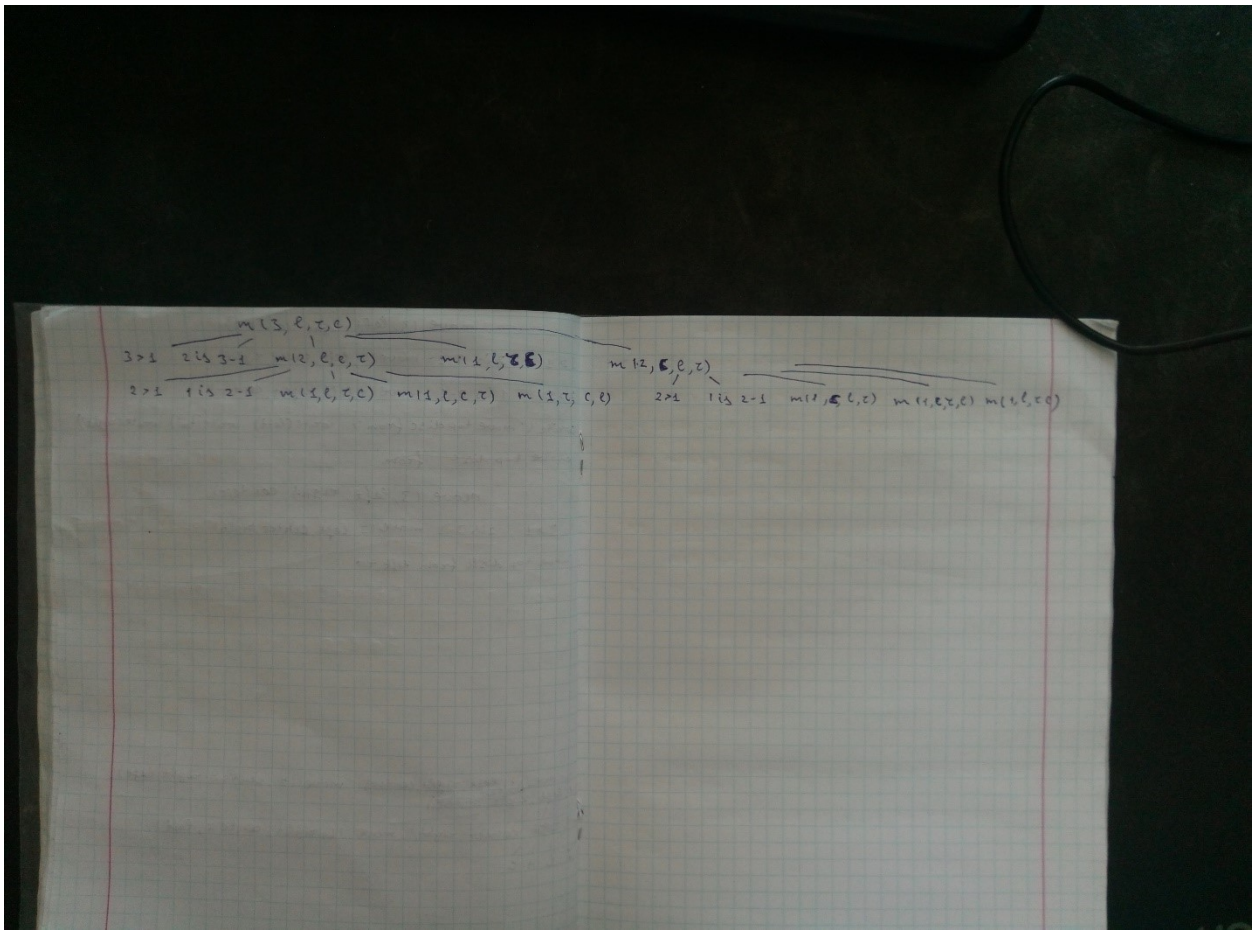
left – l

right – r

center – c

одиничний m(1,l,r,c) просто виводе на екран move top disk from left to right

якщо кожен такий m розпишу, на 4 стрілки в низ, то дерево не влізе в зошит:



Вправа 2.3.2 Пробують мету Прологу ?-move(3,left,right,left). What' неправильно? Запропонуйте дорогу виправити це і керуватися, щоб бачити, що fix роботи.

(*Упражнение 2.3.2.* Попробуйте поставить цель Пролога ?-переместить(3,влево,вправо,влево). Что случилось? Предложите способ исправить это и следуйте инструкциям, чтобы убедиться, что исправление работает.)

```
move(1,X,Y,_) :-
 write('Move top disk from '),
 write(X),
 write(' to '),
 write(Y),
 nl.
move(N,X,Y,Z) :-
 N>1,
 M is N-1,
 move(M,X,Z,Y),
 move(1,X,Y,_),
 move(M,Z,Y,X).
```

```
?- move(3,left,right,left).
Move top disk from left to right
Move top disk from left to left
Move top disk from right to left
Move top disk from left to right
Move top disk from left to left
Move top disk from left to right
Move top disk from left to right
true
```

Програма спрацювала без помилок. Але поставила однакові назви на ліво і на центр(назвала його «ліво»). Тому відпрацювала перестановки для двух різних ліво, які ми не відрізняємо по назві.

Виправити це можна, якщо через кому написати X \= Y, X \= Z, Z \= Y ось реалізація в коді:

```
move(1,X,Y,_) :-
 write('Move top disk from '),
 write(X),
 write(' to '),
 write(Y),
 nl.
move(N,X,Y,Z) :-
 N>1,
 X \= Y, X \= Z, Z \= Y,
 M is N-1,
 move(M,X,Z,Y),
 move(1,X,Y,_),
 move(M,Z,Y,X).
```

Тоді будуть аргументи з однаковими назвами стовпців давати результат false:

```
?- move(3,left,right,left).
false.

?- move(3,left,right,senter).
Move top disk from left to right
Move top disk from left to senter
Move top disk from right to senter
Move top disk from left to right
Move top disk from senter to left
Move top disk from senter to right
Move top disk from left to right
true
```