

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт
з лабораторної роботи № 5 з дисципліни
“Мультипарадигменне програмування”

Виконав студент:

ІП-01 Танасієнко Олександр

Перевірив:

ас. Очеретяний О. К.

Київ 2022

1. Завдання лабораторної роботи

Exercise 2.1 If a map has N regions, then estimate how many computations may have to be done in order to determine whether or not the coloring is in conflict. Argue using program clause trees.

Дано : мапа налічує N регіонів, підрахуйте максимальну кількість обчислень які необхідно зробити для визначення чи є конфлікт кольорів? Обґрунтуйте вашу відповідь із використанням дерева фактів програмного коду

Exercise 2.2.1 Using the first factorial program, show explicitly that there cannot possibly be an clause tree rooted at 'factorial(5,2)' having all true leaves.

Використовуючи першу програму факторіал покажіть що не існує дерева фактів що має корінь 'factorial(5,2)' що має всі листки true

Exercise 2.2.2 Draw an clause tree for the goal 'factorial(3,1,6)' having all true leaves, in a fashion similar to that done for factorial(3,6) previously. How do the two programs differ with regard to how they compute factorial? Also, trace the goal 'factorial(3,1,6)' using Prolog.

Напишіть дерево фактів для цілі 'factorial(3,1,6)' маючи всі листки true, в такому ж форматі як це зроблено для factorial(3,6) попередньо. Як ці дві програми відрізняються з точки зору підрахунку факторіалу? Також виконайте трасування цілі 'factorial(3,1,6)' використовуючи Prolog

Exercise 2.3.1 Draw a program clause tree for the goal 'move(3,left,right,center)' show that it is a consequence of the program. How is this clause tree related to the substitution process explained above?

Напишіть дерево фактів для цілі 'move(3,left,right,center)' , покажіть що це програмна послідовність.

Яким чином це дерево фактів стосується процесу підстановки описаного вище.

Exercise 2.3.2 Try the Prolog goal `?-move(3,left,right,left)`. What's wrong? Suggest a way to fix this and follow through to see that the fix works.

Виконайте ціль Prolog `move(3,left,right,left)`. У чому помилка? Запропонуйте шлях вирішення та перевірте що виправлення спрацювало.

2. Виконання роботи

Exercise 2.1

Отже, для підрахування максимальної кількості обчислень визначимо загальну кількість перевірок пар. Для визначення конфлікту кольорів нам потрібно взяти кожну пару суміжних регіонів і перевірити їх на конфлікт кольорів. У найгіршому випадку кожний регіон суміжний з усіма іншими, тобто загальна кількість перевірок буде кількістю усіх можливих пар, тобто кількість комбінацій з n по 2 елементи: $C_n^2 = \frac{n!}{(n-2)!2!} = \frac{n(n-1)}{2}$. Оскільки перевірка однієї пари це три операції:

```

1 conflict(Coloring) :-
2     adjacent(X,Y),
3     color(X,Color,Coloring),
4     color(Y,Color,Coloring).
5

```


- перевірка суміжності, отримання кольору першої вершини і перевірка, чи має друга вершина такий же колір, а кожну пару програма на Prolog перевірятиме двічі: спочатку для першої вершини перевірить чи має інший колір суміжна вершина, а потім аналогічна перевірка для другої вершини з першою, бо дві області суміжні, якщо $\text{adjacent}(v1,v2)$ і $\text{adjacent}(v2,v1)$, то загальна кількість обчислень становить: $3 * 2 \frac{n(n-1)}{2} = 3 * n(n-1)$. Впевнимось у цьому, взявши 3 вершини: 1, 2 і 3, що пофарбовані відповідно в червоний, синій і зелений:

```

1 adjacent(1,2).      adjacent(2,1).
2 adjacent(1,3).      adjacent(3,1).
3 adjacent(2,3).      adjacent(3,2).
4
5 color(1,red,a).
6 color(2,blue,a).
7 color(3,green,a).
8
9 conflict(Coloring) :-
0     adjacent(X,Y),
1     color(X,Color,Coloring),
2     color(Y,Color,Coloring).
3

```

Перевіримо наявність конфліктів:

 trace, conflict(a)

Fail. 
false

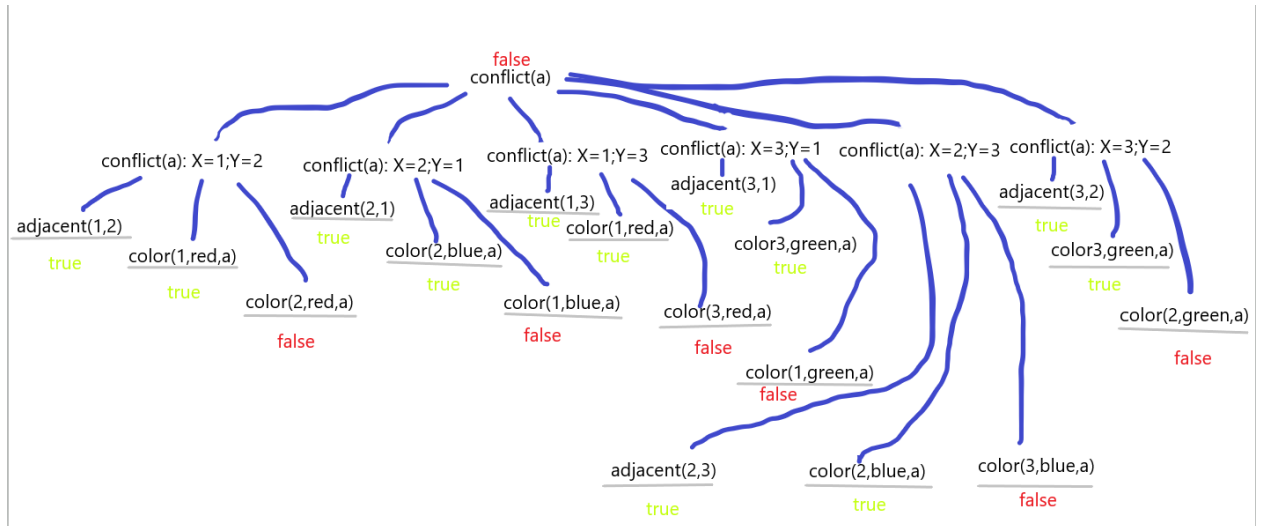
Результат:

Трасування виконання:

```
Call: conflict(a)
Call: adjacent(_666,_668)
Exit: adjacent(1,2)
Call: color(1,_670,a)
Exit: color(1,red,a)
Call: color(2,red,a)
Fail: color(2,red,a)
Redo: adjacent(_666,_668)
Exit: adjacent(2,1)
Call: color(2,_670,a)
Exit: color(2,blue,a)
Call: color(1,blue,a)
Fail: color(1,blue,a)
Redo: adjacent(_666,_668)
Exit: adjacent(1,3)
Call: color(1,_670,a)
Exit: color(1,red,a)
Call: color(3,red,a)
Fail: color(3,red,a)
Redo: adjacent(_666,_668)
Exit: adjacent(3,1)
Call: color(3,_670,a)
Exit: color(3,green,a)
Call: color(1,green,a)
Fail: color(1,green,a)
Redo: adjacent(_666,_668)
Exit: adjacent(2,3)
Call: color(2,_670,a)
Exit: color(2,blue,a)
Call: color(3,blue,a)
Fail: color(3,blue,a)
Redo: adjacent(_666,_668)
Exit: adjacent(3,2)
Call: color(3,_670,a)
Exit: color(3,green,a)
Call: color(2,green,a)
Fail: color(2,green,a)
Fail: conflict(a)
```

false

Побудуємо дерево фактів:



Можемо побачити, що загалом було проведено 18 операцій, $n=3$, $3*3*2=18$, тобто визначена вище формула є справедливою.

Exercise 2.2.1

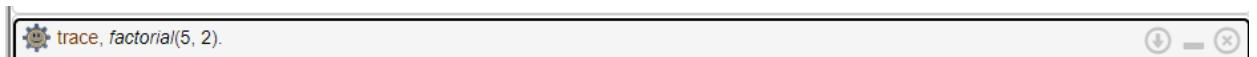
Використаємо програму:

```

1 factorial(0,1).
2
3 factorial(N,F) :-
4     N>0,
5     N1 is N-1,
6     factorial(N1,F1),
7     F is N * F1.
8

```

Отже, виконаємо трасування для `factorial(5, 2)`, щоб побачити дерево фактів:

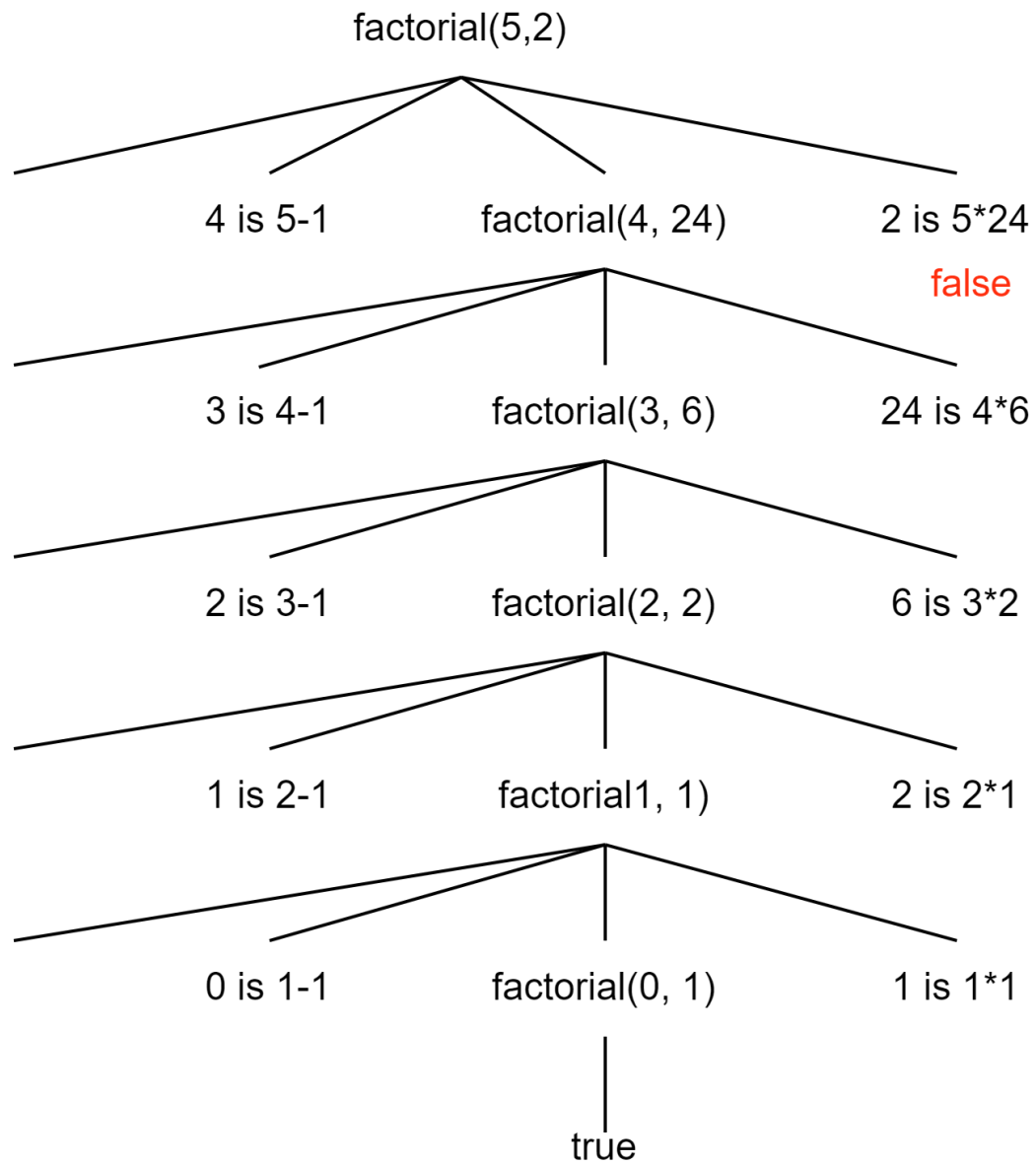


Було отримано наступний результат:

```
Call: factorial(5,2)
Call: 5>0
Exit: 5>0
Call: _630 is 5 + -1
Exit: 4 is 5 + -1
Call: factorial(4,_632)
Call: 4>0
Exit: 4>0
Call: _640 is 4 + -1
Exit: 3 is 4 + -1
Call: factorial(3,_636)
Call: 3>0
Exit: 3>0
Call: _644 is 3 + -1
Exit: 2 is 3 + -1
Call: factorial(2,_646)
Call: 2>0
Exit: 2>0
Call: _654 is 2 + -1
Exit: 1 is 2 + -1
Call: factorial(1,_656)
Call: 1>0
Exit: 1>0
Call: _664 is 1 + -1
Exit: 0 is 1 + -1
Call: factorial(0,_666)
Exit: factorial(0,1)
Call: _656 is 1*1
Exit: 1 is 1*1
Exit: factorial(1,1)
Call: _646 is 2*1
Exit: 2 is 2*1
Exit: factorial(2,2)
Call: _636 is 3*2
Exit: 6 is 3*2
Exit: factorial(3,6)
Call: _626 is 4*6
Exit: 24 is 4*6
Exit: factorial(4,24)
Call: 2 is 5*24
Fail: 2 is 5*24
Exit: factorial(5,2)
```

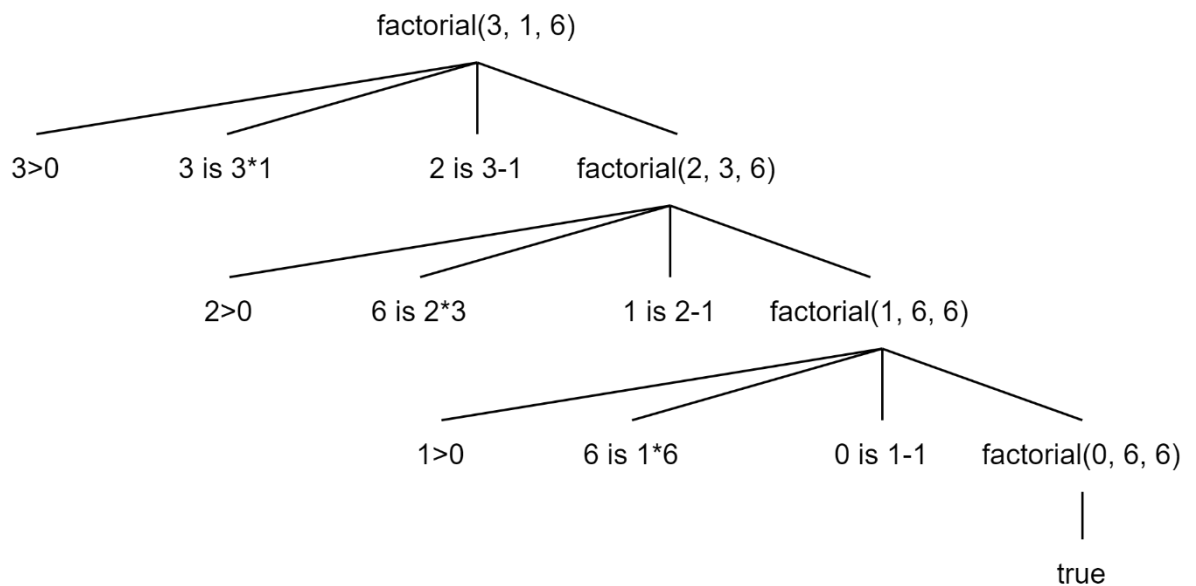
Після виконання трасування можемо явно побачити, що не існує дерева фактів, що має корінь `factorial(5,2)` і має всі листки `true`. На скріншоті трасування видно, що при перевірці 2 на рівність $5 \cdot 24$ маємо `false`, тобто

очікуваний факторіал і знайдений не співпадають, бо факторіал 5 не рівний 2.
Побудуємо дерево фактів:

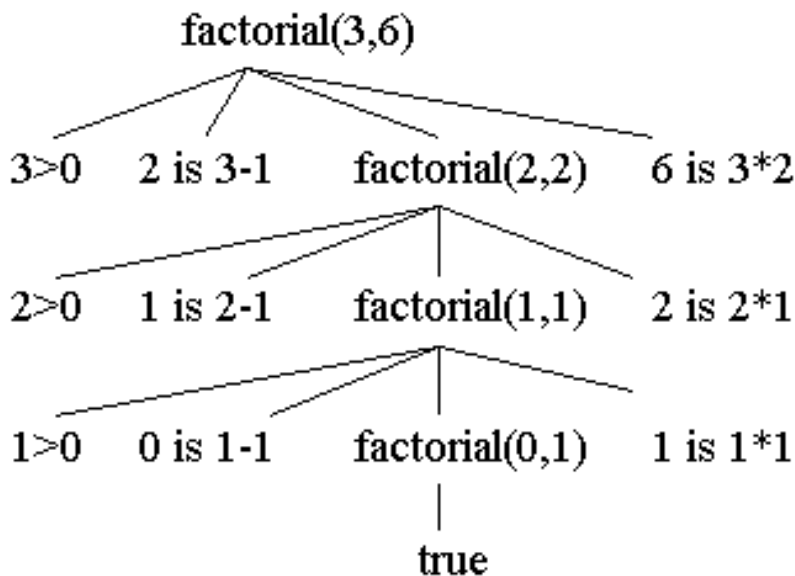


Exercise 2.2.2

Спочатку побудуємо дерево фактів для цілі `factorial(3,1,6)`:



Попередньо, для `factorial(3,6)` було отримано наступне дерево:



Різниця в цих програмах з точки зору підрахунку факторіалу:


Початкова програма, для якої робився виклик `factorial(3,6)`, виконувала порівняння отриманого значення факторіала і заданого біля кореня. Факторіал для $n-1$ було обчислено при рекурсивному заглибленні і повернуто нагору до листка біля кореня. Друга програма, навпаки, виконувала обчислення під час рекурсивного заглиблення, перевірка заданого та отриманого значення факторіалу відбувалася у листку на

найглибшому рівні, а у корінь поверталось вже значення true/false.
Виконаємо трасування цілі factorial(3,1,6), використовуючи Prolog:

```
1 factorial(0,F,F).  
2  
3 factorial(N,A,F) :-  
4     N > 0,  
5     A1 is N*A,  
6     N1 is N -1,  
7     factorial(N1,A1,F).
```

 factorial(3,1,6)
true

Трасування:

 trace, factorial(3,1,6)

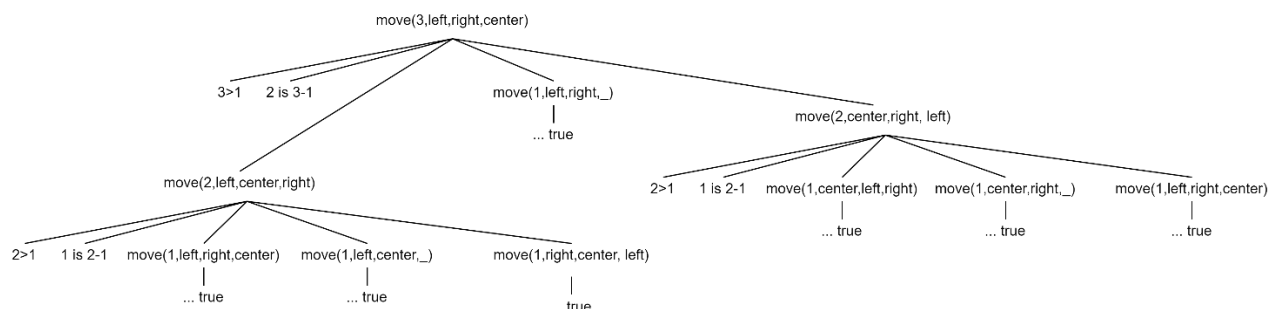
```
Call: factorial(3,1,6)  
Call: 3>0  
Exit: 3>0  
Call: _630 is 3*1  
Exit: 3 is 3*1  
Call: _644 is 3 + -1  
Exit: 2 is 3 + -1  
Call: factorial(2,3,6)  
Call: 2>0  
Exit: 2>0  
Call: _646 is 2*3  
Exit: 6 is 2*3  
Call: _648 is 2 + -1  
Exit: 1 is 2 + -1  
Call: factorial(1,6,6)  
Call: 1>0  
Exit: 1>0  
Call: _650 is 1*6  
Exit: 6 is 1*6  
Call: _664 is 1 + -1  
Exit: 0 is 1 + -1  
Call: factorial(0,6,6)  
Exit: factorial(0,6,6)  
Exit: factorial(1,6,6)  
Exit: factorial(2,3,6)  
Exit: factorial(3,1,6)  
true
```

Exercise 2.3.1

```
1 move(1,X,Y,_) :-
2   write('Move top disk from '),
3   write(X),
4   write(' to '),
5   write(Y),
6   nl.
7 move(N,X,Y,Z) :-
8   N>1,
9   M is N-1,
10  move(M,X,Z,Y),
11  move(1,X,Y,_),
12  move(M,Z,Y,X).
```

move(3,left,right,center).

Побудуємо дерево фактів. Введемо наступні позначення для зручності: l – left, r – right, c – center.



...true – не розписував цю частину бо вона відповідає тільки за виведення даних на консоль і не показує роботу алгоритму.

`trace,move(3,left,right,center)`

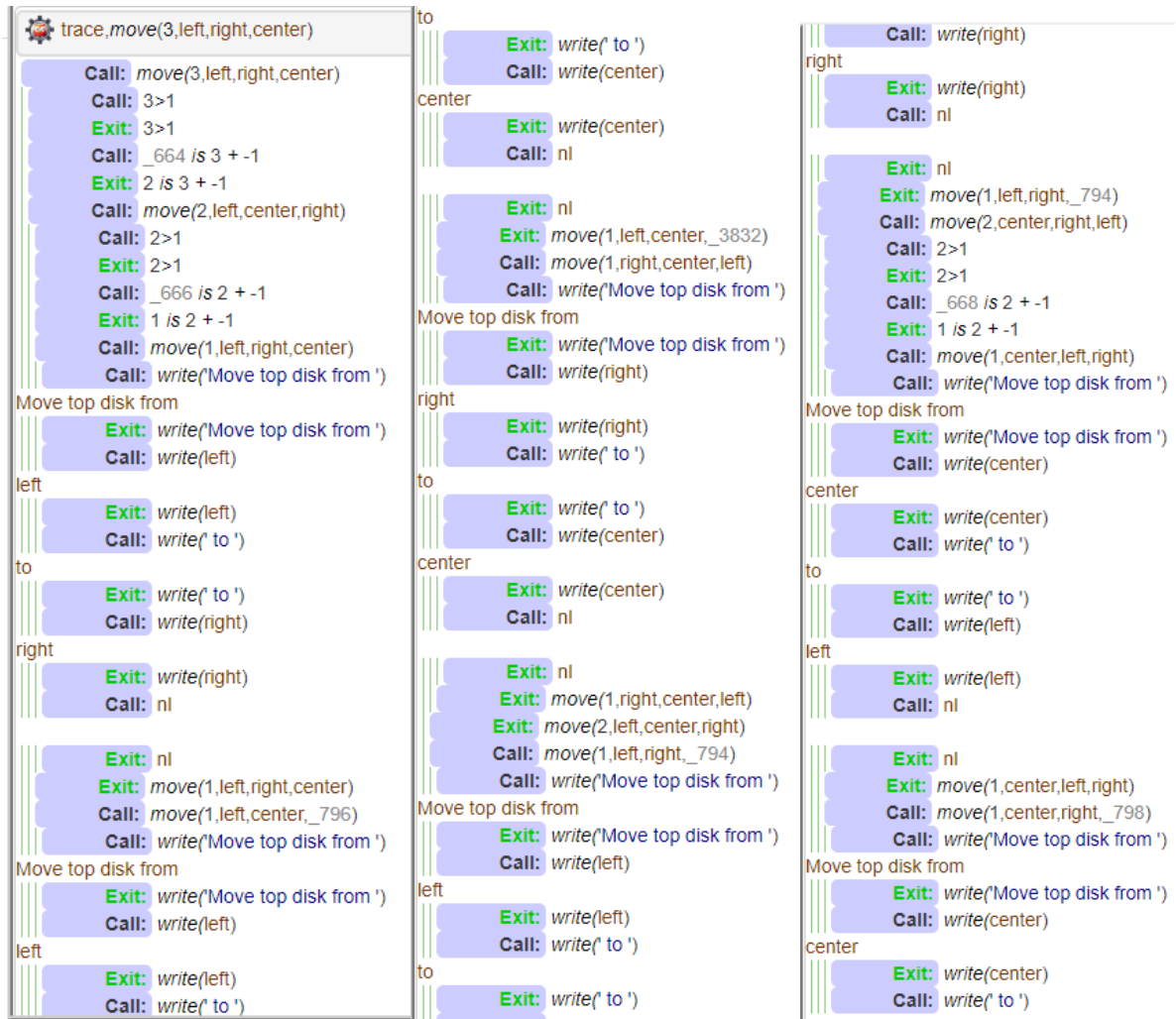
Call: `move(3,left,right,center)`

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

Exit: `move(3,left,right,center)`

`true`

Трасування виконання:



```

Call: write(" to ")
to
Exit: write(" to ")
Call: write(right)
right
Exit: write(right)
Call: nl
Exit: nl
Exit: move(1, center, right, _3834)
Call: move(1, left, right, center)
Call: write("Move top disk from ")
Move top disk from
Exit: write("Move top disk from ")
Call: write(left)
left
Exit: write(left)
Call: write(" to ")
to
Exit: write(" to ")
Call: write(right)
right
Exit: write(right)
Call: nl
Exit: nl
Exit: move(1, left, right, center)
Exit: move(2, center, right, left)
Exit: move(3, left, right, center)
true

```

Отже, можемо впевнитися, що маємо програмну послідовність. Це дерево збігається з процесом підстановки, описаним вище, та є його підтвердженням.

Exercise 2.3.2

Виконаємо: `move(3,left,right,left)`:



```
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 створила рішення і ми можемо його побачити. Проте помилка полягає в тому, що ми маємо третю вісь таку ж, як і першу, і наша програма не оброблює це ніяким чином. Зокрема, деколи програма пропонує рухати диск з лівої осі на ліву ж. Тому створене рішення є некоректним. Для вирішення проблеми додамо перевірки до коду програми, щоб на вхід передавалися різні осі:

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

Тепер при таких вже вхідних параметрах маємо false:



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