Lab Report 3 Date:2081/05/29

Title: Recursion in Prolog

1. Objective: The objective of this lab is to understand the concept of recursion in Prolog, implement recursion in Prolog and demonstrate how recursion is used to solve problems by writing two program:

* A recursive program to compute the factorial of a number, where the number is read from the user.
* A recursive solution to Factorial Number and the Tower of Hanoi (TOH) problem.

1. Theory:

**In prolog,** Recursion is used when a problem can be broken down into smaller sub-problems of the same type. Each recursive definition consists of:

* Base Case: The condition that stops the recursion, providing a direct answer.
* Recursive Case: The rule that calls the predicate itself with modified arguments, reducing the problem size step by step.

2.1 Factorial:

The factorial of a number N, denoted as N!, is the product of all positive integers up to N. The recursive definition is:

* 0! = 1 (base case)
* N! = N \* (N-1)! for N > 0 (recursive case)

2.2 Tower of Hanoi:

The TOH problem involves moving a stack of disks from one peg to another,

using an auxiliary peg, under the constraints that only one disk can be moved at a time, and no larger disk can be placed on a smaller disk. The recursive solution involves:

*  Moving N-1 disks from the source peg to the auxiliary peg.
*  Moving the largest disk directly to the destination peg.
*  Moving the N-1 disks from the auxiliary peg to the destination peg.

1. Recursive program to find the factorial of a number.

*% Base case for factorial: 0! = 1*

*factorial(0, 1).*

*% Recursive case for factorial: N! = N \* (N-1)!*

*factorial(N, Result) :-*

*N > 0,*

*N1 is N - 1,*

*factorial(N1, Result1),*

*Result is N \* Result1.*

*% Main entry point to get user input and print factorial*

*main :-*

*write('Enter a number: '),*

*read(Number),*

*factorial(Number, Result),*

*write('Factorial of '), write(Number), write(' is '), write(Result), nl.*

Output:



1. Recursive program of Tower of Hanoi.

*% Base case: moving one disk from the source to the destination peg*

*hanoi(1, Source, Destination, \_) :-*

*write('Move disk 1 from '), write(Source), write(' to '), write(Destination), nl.*

*% Recursive case: moving N disks from source to destination using auxiliary peg*

*hanoi(N, Source, Destination, Auxiliary) :-*

*N > 1,*

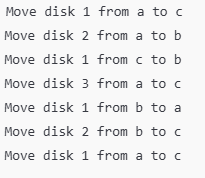
*M is N - 1,*

*hanoi(M, Source, Auxiliary, Destination), % Move N-1 disks from source to auxiliary*

*write('Move disk '), write(N), write(' from '), write(Source), write(' to '), write(Destination), nl,*

*hanoi(M, Auxiliary, Destination, Source). % Move N-1 disks from auxiliary to destination*

Output of Hanoi(3,a,c,b):



1. Conclusion:

This lab demonstrates the use of **recursion** in Prolog to solve classic problems. The **factorial** calculation and the **Tower of Hanoi** puzzle both showcase how recursion can be applied to break down complex tasks into smaller, manageable subproblems. Through these examples, we can see the elegance and efficiency of recursive solutions in Prolog, where each solution step is naturally expressed in terms of smaller subgoals.