**Lab # 5**

Due: 2024-10-24

By: Omar Abdul, Anas Taimah, Yusuf Khan

300228700, 300228842, 300293842

Course: CSI3120

Group: 40

**Code explanation:**

Task 1:

**Facts of the family tree**: The first part of this code was provided within the lab instructions, and its purpose it to provide the facts of the family tree itself. This includes the facts of the parent-child relationship and the genders of the family tree.

**Rule 1- Siblings:** As defined within the lab instructions, siblings are individuals who share at least one parent and are not the same person. This is done in the following manner:

sibling(X, Y) :-

parent(P, X) checks if P is a parent of X.

parent(P, Y) checks if P is also a parent of Y.

X \= Y ensures that X and Y are not the same person.

**Rule 2- Grandparents**: As defined within the lab instructions, we can determine this by identifying if one person is a grandparent of another by chaining the `parent/2` predicate. His is done in the following manner:

grandparent(GP, GC) :-

parent(GP, P) checks if GP is a parent of P.

parent(P, GC) checks if P is a parent of GC.

With GP implying grandparent and GC implying GC, this rule checks if GC has a parent who is the child of GP, therefore showing the grandparent relationship.

**Rule 3- Ancestor**: As defined within the lab instructions, this rule first checks if a person is a direct parent, then recursively checks if they are an ancestor through other descendants.

There are two parts to this rule,

The first is:

ancestor(Anc, Desc) :-

parent(Anc, Desc).

This is the base case which check if Anc is a direct parent of Desc.

The second is the recursive case:

ancestor(Anc, Desc) :-

parent(Anc, P),

ancestor(P, Desc).

This checks if Anc is a parent of P, and then recursively checks if P is an ancestor of Desc.

Task 2:

Base Case: sum\_odd\_numbers([], 0) simply handles the case when the list is empty, and accordingly assigns it a 0.

Recursive Case #1: This is to handle the case when the head element is odd. It does the check on the head (if odd or even) by checking if the result of head modulo 2 equals 1. If this case is met, then the program recursively calculates the sum of the odd numbers in the tail the tail, with the result being store in TailSum. After this, it adds the head value to TailSum, therefore giving us the overall Sum

Recursive Case #2: This is to handle the case when the head element is even. It does the check on the head (if odd or even) by checking if the result of head modulo 2 equals 0. If the head is even, then it just skips the head and recursively runs on the tail.

Task 3:

permutation([red, blue, green, yellow], Houses): The first part of task 3 is simply using the built-in Prolog predicate “permutation” to generate all possible permutations (orderings) of the list [red, blue, green, yellow]. Each permutation is assigned to Houses, representing the possible order of house colors.

immediately\_left\_of(red, blue, Houses): This calls the custom predicate immediately\_left\_of (this is later explained) to ensure that the red house is directly to the left of the blue house in the Houses list.

nth1(GreenIndex, Houses, green): Uses the built in predicate “nth” which retrieves the index of the green house in the Houses list, with GreenIndex holding the index of the green house.

not\_next\_to(yellow, green, Houses): Calls the custom predicate “not\_next\_to”, which ensures that the yellow house and the green house are not next to each other in the Houses list.

GreenIndex \= 2: This ensures that GreenIndex, the position of the green house, is not equal to 2, meaning the green house can't be in the second position.

immediately\_left\_of(Left, Right, List): This is a custom predicate that uses append to check if Left is immediately to the left of Right in the list List

next\_to(X, Y, List): This is a custom predicate uses append to check if X and Y are next to each other in the list List. It does this by trying both orders of X before Y and Y before X.

Challenges: Initially I was not getting the correct output, which eventually required me to add the immediately\_left\_of predicate, with next\_to only ending up serving to be a helper later in not\_next\_to.

**Test cases:**

Task 1:

A screenshot of a computer code

Description automatically generated

Task 2:

A white background with black text

Description automatically generated

Task 3:

|  |
| --- |
| A screenshot of a computer code  Description automatically generated |

All of these test cases match the expected result

**ChatGPT declaration:** Our group used ChatGPT throughout the lab to better understand the program application so that we are able to develop our answers. ChatGPT was used for assistance to learn and understand how it worked and not to generate any answers or results. All solutions were implemented in our own words and approach.