Artificial Intelligence Laboratory

[CS 4271]

Assignment 2

Prolog Programs

Submitted by

Sriparno Ganguly

Enrol No. – 2020CSB004

Problem 1: Duplicating Elements of a List

Problem Description:
Write a Prolog program to duplicate the elements of a list a given number of times.

Code:

```
%%% Problem 1 %%%
% Base case: Duplicating an empty list results in an empty list
duplicate([], _, []).

% Duplicate the elements of the list N times
duplicate([H|T], N, Result) :-
    duplicate_element(H, N, Duplicated),
    duplicate(T, N, RestDuplicated),
    append(Duplicated, RestDuplicated, Result).

% Helper predicate to duplicate a single element N times
duplicate_element(_, 0, []).
duplicate_element(X, N, [X|Rest]) :-
    N > 0,
    N1 is N - 1,
    duplicate_element(X, N1, Rest).
```

Sample Query Runs:

```
?- duplicate([a,b,c], 3, X).
% X = [a, a, a, b, b, b, c, c, c]

?- duplicate([1,2,3], 2, Y).
% Y = [1, 1, 2, 2, 3, 3]
```

Problem 2: Determining Sublists

Problem Description:
Write a Prolog program to determine whether a list is a sublist of another list. A list is considered a sublist if its elements are present in another list consecutively and in the same order.

Code:

```
%%% Problem 2 %%%
% Base case: An empty list is always a sublist
is_sublist([], _).

% Predicate to check if L1 is a sublist of L2
is_sublist([X|Xs], [X|Ys]) :- % Match the first element
    is_sublist_helper(Xs, Ys). % Check the rest

is_sublist(L, [_|Ys]) :- % Skip the first element of the second list
    is_sublist(L, Ys).

% Helper predicate to check if the remaining elements of L1 are a sublist
of L2
is_sublist_helper([], _).
is_sublist_helper([X|Xs], [X|Ys]) :-
    is_sublist_helper(Xs, Ys).
```

Sample Query Runs:

```
?- is_sublist([c, d], [a, b, c, d, e]).
% true

?- is_sublist([2, 3], [1, 2, 3, 4, 5]).
% false
```

Problem 3: Set Operations

Problem Description:
Write a Prolog program to determine the intersection, union, difference, and symmetric difference of two sets.

Code:

```
%%% Problem 3 %%%
% Helper predicate to check if an element is a member of a list
my_member(X, [X|_]).
my_member(X, [_|T]) :-
    my_member(X, T).

% Intersection of two sets
intersection_set([], _, []).
intersection_set([X|Set1], Set2, Intersection) :-
    my_member(X, Set2),
    intersection_set(Set1, Set2, RestIntersection),
    Intersection = [X|RestIntersection].
intersection_set([_|Set1], Set2, Intersection) :-
    intersection_set(Set1, Set2, Intersection).

% Helper predicate to remove duplicates from a list
remove_duplicates([], []).
remove_duplicates([X|Xs], Result) :-
    member(X, Xs),
    remove_duplicates(Xs, Result).
remove_duplicates([X|Xs], [X|Result]) :-
    \+ member(X, Xs),
    remove_duplicates(Xs, Result).

% Union of two sets
union_set(Set1, Set2, Union) :-
    append(Set1, Set2, CombinedSet),
    remove_duplicates(CombinedSet, Union).

% Difference of two sets
difference_set([], _, []).
difference_set([X|Set1], Set2, Difference) :-
    \+ my_member(X, Set2),
    difference_set(Set1, Set2, RestDifference),
    Difference = [X|RestDifference].
difference_set([_|Set1], Set2, Difference) :-
    difference_set(Set1, Set2, Difference).

% Symmetric difference of two sets
symmetric_difference_set(Set1, Set2, SymmetricDifference) :-
    difference_set(Set1, Set2, Difference1),
```

```
    difference_set(Set2, Set1, Difference2),
    append(Difference1, Difference2, SymmetricDifference).
```

Sample Query Runs:

```
?- intersection_set([1, 2, 3], [2, 3, 4], Intersection).
% Intersection = [2, 3]

?- union_set([1, 2, 3], [3, 4, 5], Union).
% Union = [1, 2, 3, 4, 5]

?- difference_set([1, 2, 3, 4], [2, 3], Difference).
% Difference = [1, 4]

?- symmetric_difference_set([1, 2, 3], [2, 3, 4], SymmetricDifference).
% SymmetricDifference = [1, 4]
```

Problem 4: Transposing Lists into Pairs

Problem Description:
Write a Prolog program to transpose two lists into a list of pairs. Given two lists, L1 and L2, create pairs from corresponding elements.

Code:

```
%%% Problem 4 %%%
% Helper predicate to create pairs from corresponding elements of two
lists
pair_elements(X, Y, (X, Y)).

% Transpose lists L1 and L2 into a list of pairs L
transpose_lists(L1, L2, L) :-
    maplist(pair_elements, L1, L2, L).
```

Sample Query Runs:

```
?- transpose_lists([a, b, c], [1, 2, 3], TransposedList).
% TransposedList = [(a, 1), (b, 2), (c, 3)]

?- transpose_lists([apple, banana, cherry], [red, yellow, red], Pairs).
% Pairs = [(apple, red), (banana, yellow), (cherry, red)]
```

Problem 5: Splitting a List

Problem Description:
Write a Prolog program to split a list into two parts. The length of the first part is given.

Code:

```
%%% Problem 5 %%%
% Helper predicate to split a list into two parts
split_helper(0, L, [], L).
split_helper(N, [X|Rest], [X|L1], L2) :-
    N > 0,
    N1 is N - 1,
    split_helper(N1, Rest, L1, L2).

split(List, N, L1, L2) :-
    length(List, Len),
    between(0, Len, N),
    split_helper(N, List, L1, L2).
```

Sample Query Runs:

```
?- split([a, b, c, d, e, f, g, h, i, j, k], 3, L1, L2).
% L1 = [a, b, c], L2 = [d, e, f, g, h, i, j, k]

?- split([1, 2, 3, 4, 5], 2, FirstPart, SecondPart).
% FirstPart = [1, 2], SecondPart = [3, 4, 5]
```

Problem 6: Extracting a Slice from a List

Problem Description:
Write a Prolog program to extract a slice from a list. Given two indices, I and K, the slice is the list containing the elements between the Ith and Kth element of the original list (both limits included). Start counting the elements with 1.

Code:

```
%%% Problem 6 %%%
slice([X|_], 1, 1, [X]).

slice([X|Xs], 1, K, [X|Ys]) :-
    K > 1,
    K1 is K - 1,
    slice(Xs, 1, K1, Ys).

slice([_|Xs], I, K, Ys) :-
    I > 1,
    I1 is I - 1,
    K1 is K - 1,
    slice(Xs, I1, K1, Ys).
```

Sample Query Runs:

```
?- slice([a, b, c, d, e, f, g, h, i, j, k], 3, 7, L).
% L = [c, d, e, f, g]

?- slice([1, 2, 3, 4, 5], 2, 4, Result).
% Result = [2, 3, 4]
```

Problem 7: Generating Combinations

Problem Description:
Write a Prolog program to generate combinations of K distinct objects chosen from the N elements of a list. In how many ways can a committee of K be chosen from a group of N people? The number of possibilities is given by C(N, K), where C(N, K) denotes the well-known binomial coefficients.

Code:

```
%%% Problem 7 %%%
% Helper predicate to select K elements from a list
selectK(0, _, []).
selectK(K, [X|Xs], [X|Ys]) :-
    K > 0,
    K1 is K - 1,
    selectK(K1, Xs, Ys).
selectK(K, [_|Xs], Ys) :-
    K > 0,
    selectK(K, Xs, Ys).

% Main predicate to generate combinations
combinations(K, List, Result) :-
    selectK(K, List, Result).
```

Sample Query Runs:

```
?- combinations(3, [a, b, c, d, e, f], L).
% L = [a, b, c]; L = [a, b, d]; L = [a, b, e]; ... (and so on)

?- combinations(2, [1, 2, 3, 4], Combo).
% Combo = [1, 2]; Combo = [1, 3]; Combo = [1, 4]; ... (and so on)
```

Problem 8: Sorting Algorithms

Problem Description:
Implement Bubble Sort, Insertion Sort, and Merge Sort algorithms in Prolog.

Code:

```
%%% Problem 8 %%%
% Bubble Sort %
% Helper predicate to swap two elements in a list
swap([X, Y|Rest], [Y, X|Rest]) :- X > Y.
swap([Z|Rest1], [Z|Rest2]) :- swap(Rest1, Rest2).

% Base case: An already sorted list
bubble_sort(List, List) :- \+ (swap(List, NewList), List \= NewList).

% Recursive case: Continue sorting
bubble_sort(List, Sorted) :- swap(List, Swapped), bubble_sort(Swapped,
Sorted).

% Insertion Sort %
% Helper predicate to insert an element into a sorted list
insert(X, [], [X]).
insert(X, [Y|Rest], [X,Y|Rest]) :- X =< Y.
insert(X, [Y|Rest1], [Y|Rest2]) :- X > Y, insert(X, Rest1, Rest2).

% Base case: An empty list is already sorted
insertion_sort([], []).

% Recursive case: Insert the head into the sorted tail
insertion_sort([H|T], Sorted) :- insertion_sort(T, SortedTail), insert(H,
SortedTail, Sorted).

% Merge Sort %
% Helper predicate to merge two sorted lists
merge([], L, L).
merge(L, [], L).
merge([X|Xs], [Y|Ys], [X|Z]) :- X =< Y, merge(Xs, [Y|Ys], Z).
merge([X|Xs], [Y|Ys], [Y|Z]) :- X > Y, merge([X|Xs], Ys, Z).

% Base case: An empty list or a single-element list is already sorted
merge_sort([], []).
merge_sort([X], [X]).

% Recursive case: Split the list into two, sort each part, and merge them
merge_sort(List, Sorted) :-
    length(List, Len),
    Len > 1,
    HalfLen is Len // 2,
```

```
    length(Left, HalfLen),
    append(Left, Right, List),
    merge_sort(Left, SortedLeft),
    merge_sort(Right, SortedRight),
    merge(SortedLeft, SortedRight, Sorted).
```

Sample Query Runs:

```
?- bubble_sort([5, 3, 1, 4, 2], Sorted).
% Sorted = [1, 2, 3, 4, 5]

?- insertion_sort([8, 2, 7, 1, 4], Sorted).
% Sorted = [1, 2, 4, 7, 8]

?- merge_sort([10, 5, 8, 2, 7], Sorted).
% Sorted = [2, 5, 7, 8, 10]
```

Problem 9: Packing and Encoding Consecutive Duplicates

Problem Description:
Implement Prolog predicates to:

1. Pack consecutive duplicates of list elements into sublists. If a list contains repeated elements, they should be placed in separate sublists.
2. Encode consecutive duplicates as terms [N, E] where N is the number of duplicates of the element E.

Code:

```
%%% Problem 9 %%%
% Helper predicate for 'pack/2'
pack_helper([], []).
pack_helper([X], [[X]]).
pack_helper([X, X | T], [[X | Rest] | RestPacked]) :-
    pack_helper([X | T], [Rest | RestPacked]).
pack_helper([X, Y | T], [[X] | Packed]) :-
    \+ X = Y,
    pack_helper([Y | T], Packed).

% Predicate to pack consecutive duplicates into sublists
pack(List, PackedList) :-
    pack_helper(List, PackedList).

% Helper predicate for 'encode/2'
encode_helper([], []).
encode_helper([X], [[1, X]]).
encode_helper([X, X | T], [[N, X] | RestEncoded]) :-
    encode_helper([X | T], [[M, X] | RestEncoded]),
    N is M + 1.
encode_helper([X, Y | T], [[1, X] | Encoded]) :-
    \+ X = Y,
    encode_helper([Y | T], Encoded).

% Predicate to encode consecutive duplicates as terms [N, E]
encode(List, EncodedList) :-
    encode_helper(List, EncodedList).
```

Sample Query Runs:

```
?- pack([a, a, a, a, b, c, c, a, a, d, e, e, e, e], Packed).
% Packed = [[a, a, a, a], [b], [c, c], [a, a], [d], [e, e, e, e]]

?- encode([a, a, a, a, b, c, c, a, a, d, e, e, e, e], Encoded).
% Encoded = [[4, a], [1, b], [2, c], [2, a], [1, d], [4, e]]
```

Problem 10: Smoothie Store Information

Store Information:

```
store(best_smoothies, [alan,john,mary],
    [ smoothie(berry, [orange, blueberry, strawberry], 2),
      smoothie(tropical, [orange, banana, mango, guava], 3),
      smoothie(blue, [banana, blueberry], 3) ]).

store(all_smoothies, [keith,mary],
    [ smoothie(pinacolada, [orange, pineapple, coconut], 2),
      smoothie(green, [orange, banana, kiwi], 5),
      smoothie(purple, [orange, blueberry, strawberry], 2),
      smoothie(smooth, [orange, banana, mango],1) ]).

store(smoothies_galore, [heath,john,michelle],
    [ smoothie(combo1, [strawberry, orange, banana], 2),
      smoothie(combo2, [banana, orange], 5),
      smoothie(combo3, [orange, peach, banana], 2),
      smoothie(combo4, [guava, mango, papaya, orange],1),
      smoothie(combo5, [grapefruit, banana, pear],1) ]).
```

Sub-Problem-a: Stores with More Than Four Smoothies

```
more_than_four(X) :-
    store(X, _, Smoothies),
    length(Smoothies, N),
    N ⩾ 4.
```

Sample Query Run:

```
?- more_than_four(X).
% X = best_smoothies ; X = all_smoothies ; X = smoothies_galore
```

Sub-Problem-b: Check if a Smoothie Exists

```
exists(X) :-
    store(_, _, Smoothies),
    member(smoothie(X, _, _), Smoothies).
```

Sample Query Run:

```
?- exists(green).
% true
```

Sub-Problem-c: Employee-to-Smoothie Ratio

```
calculate_ratio(Store, Ratio) :-
    store(Store, Employees, Smoothies),
    length(Employees, NumEmployees),
    length(Smoothies, NumSmoothies),
    NumSmoothies > 0, % Avoid division by zero
    Ratio is NumEmployees / NumSmoothies.

ratio(X, R) :-
    calculate_ratio(X, R).
```

Sample Query Run:

```
?- ratio(smoothies_galore, R).
% R = 0.6
```

Sub-Problem-d: Average Smoothie Price

```
calculate_total_price([], 0).
calculate_total_price([smoothie(_, _, Price) | Rest], Total) :-
    calculate_total_price(Rest, RemainingTotal),
    Total is RemainingTotal + Price.

calculate_num_smoothies([], 0).
calculate_num_smoothies([_|Rest], Num) :-
    calculate_num_smoothies(Rest, RemainingNum),
    Num is RemainingNum + 1.

average(X, A) :-
    store(X, _, Smoothies),
    calculate_total_price(Smoothies, TotalPrice),
    calculate_num_smoothies(Smoothies, NumSmoothies),
    NumSmoothies > 0, % Avoid division by zero
    A is TotalPrice / NumSmoothies.
```

Sample Query Run:

```
?- average(all_smoothies, A).
% A = 2.5
```

Sub-Problem-e: Check if a Fruit is an Ingredient in All Smoothies

```
fruit_in_smoothie(Fruit, smoothie(_, Ingredients, _)) :-
    member(Fruit, Ingredients).

fruit_in_all_smoothies_helper(_, []).
fruit_in_all_smoothies_helper(Fruit, [Smoothie | RestSmoothies]) :-
```

```
    fruit_in_smoothie(Fruit, Smoothie),
    fruit_in_all_smoothies_helper(Fruit, RestSmoothies).

fruit_in_all_smoothies(Store, Fruit) :-
    store(Store, _, Smoothies),
    fruit_in_all_smoothies_helper(Fruit, Smoothies).
```

Sample Query Run:

```
?- fruit_in_all_smoothies(smoothies_galore, banana).
% true
```
