Prolog Assignment 1

Giovanni Filomeno, Manu Gupta

Exercise 1: On Arithmetic Operations in Prolog

```prolog
% Operator definitions with evaluation
apply_op(add, X, Y, Z) :- Z is X + Y.
apply_op(sub, X, Y, Z) :- Z is X - Y.
apply_op(mul, X, Y, Z) :- Z is X * Y.
apply_op(div, X, Y, Z) :- Y \= 0, Z is X / Y.

% solve/3 - main predicate
solve(L, N,Ops) :-
    solve_helper(L, N, [], Ops).

% solve_helper/4 - helper predicate
solve_helper([N], N, Ops, Ops).
solve_helper([X,Y|Rest], N, AccOps, Ops) :-
    apply_op(Op, X, Y, Z),
    append(AccOps, [[0,Op]], NewAccOps),
    solve_helper([Z|Rest], N, NewAccOps, Ops).

% Example tests
test_solve :-
    solve([8,2,3,6,2], 27, Ops1), writeln(Ops1).
    % solve([1, 2, 3, 4], 10, Ops2), writeln(Ops2),
    % solve([5, 2, 4, 8], 1, Ops3), writeln(Ops3),
    % solve([10, 2, 5, 2], 40, Ops4), writeln(Ops4).

% Query to run tests
:- test_solve.
```

Overview: Develop a Prolog predicate, **solve(L, N), that generates a sequence of arithmetic operations on a list L to match a target number N.** The sequence of operations and list transformations at each step are displayed.

Objective: The goal is to recursively explore arithmetic operations between consecutive numbers in L, testing each combination's ability to achieve N.

Approach: The Prolog program defines and applies operations using apply_op, then recursively processes these using solve_helper. It accumulates and backtracks operations to reach N.

Exercise 1: On Arithmetic Operations in Prolog

solve([8,2,3,6,2], 27).

Output: [[0,sub],[0,add],[0,mul],[0,div]]

The solution is equivalent to the number expression ((8-2)+3)*6/2).

```
[[0,sub],[0,add],[0,mul],[0,div]]
true.

?- |
```

Exercise 2: Prolog Compression & Decompression

```prolog
% Define the predicate compress/2
compress(List, CompressedList) :-
    compress_helper(List, CompressedList, 1).

% Helper predicates for compression
compress_helper([], [], _).
compress_helper([X], [[X, Count]], Count) :- Count > 2.
compress_helper([X], [X, X], 2).
compress_helper([X], [X], 1).
compress_helper([X, Y | Rest], Compressed, Count) :-
    X \= Y,
    (   Count > 2
    ->  Compressed = [[X, Count] | RestCompressed]
    ;   Count = 2
    ->  Compressed = [X, X | RestCompressed]
    ;   Compressed = [X | RestCompressed]),
    compress_helper([Y | Rest], RestCompressed, 1).
compress_helper([X, X | Rest], Compressed, Count) :-
    NewCount is Count + 1,
    compress_helper([X | Rest], Compressed, NewCount).

% Define the predicate decompress/2
decompress(CompressedList, List) :-
    decompress_helper(CompressedList, List).

% Helper predicate for decompression
decompress_helper([], []).
decompress_helper([[N, C] | T], List) :-
    C > 2,
    length(Full, C),
    maplist(=(N), Full),
    decompress_helper(T, Rest),
    append(Full, Rest, List).
decompress_helper([X | T], [X | List]) :-
    decompress_helper(T, List).

% Test predicates
test_compress :-
    compress([2,2,2,3,3,3,4,4,5,5,5,5,6,6,6,6,6], CompressedList),
    writeln('Compressed:'), writeln(CompressedList).

test_decompress :-
    decompress([[2, 3], [3, 3], 4, 4, [5, 4], [6, 5]], DecompressedList),
    writeln('Decompressed:'), writeln(DecompressedList).

% Execute tests
:- initialization(test_compress).
:- initialization(test_decompress).
```

Overview: Write Prolog predicates compress and decompress to handle lists. compress encodes segments of identical numbers, and decompress restores the original list structure.

Compression: The compression algorithm encodes consecutive identical numbers: segments longer than two are encoded as [number, length], shorter segments remain unchanged.

Decompression: The decompression process reverses the compression encoding, expanding each segment based on stored values to reconstruct the original list.

Exercise 2: Prolog Compression & Decompression

Example:

compress([2,2,2,3,3,3,4,4,5,5,5,5,6,6,6,6,6], CompressedList).

decompress([[2, 3], [3, 3], 4, 4, [5, 4], [6, 5]], L).

Output:

```
Compressed:
[[2,3],[3,3],4,4,[5,4],[6,5]]
Decompressed:
[2,2,2,3,3,3,4,4,5,5,5,5,6,6,6,6,6]
true.

?- |
```

Exercise 3: Advanced Compression Techniques

```prolog
% Helper predicate to handle the compression of sequences
compress_helper([], [], _Start, _End, _LastAdded).
compress_helper([H|T], Compressed, Start, End, LastAdded) :-
    Next is End + 1,
    (   H == Next
    ->  % Continue the current sequence
        compress_helper(T, Compressed, Start, H, LastAdded)
    ;   % Sequence breaks, decide on compression
        Length is End - Start + 1,
        (   Length > 2
        ->  NewSegment = [[Start, End]]
        ;   Length == 2
        ->  NewSegment = [Start, End]
        ;   NewSegment = [Start]
        ),
        % Recursive call with new start, reset end
        compress_helper(T, Rest, H, H, H),
        % Append the new segment to the result
        append(NewSegment, Rest, Compressed)
    ).


% Public interface for compression
compress([H|T], Compressed) :-
    compress_helper(T, Compressed, H, H, H).
compress([], []).

% Initialization for an empty list compression
compress([], []).

% Helper predicate for expanding range [Start, End]
expand_range(Start, End, List) :-
    findall(X, between(Start, End, X), List).

% Decompressing the compressed list
decompress_helper([], []).
decompress_helper([H|T], List) :-
    (   is_list(H)
    ->  H = [Start, End],
        expand_range(Start, End, Expanded),
        decompress_helper(T, Rest),
        append(Expanded, Rest, List)
    ;   decompress_helper(T, Rest),
        List = [H|Rest]
    ).

% Public interface for decompression
decompress(Compressed, List) :-
    decompress_helper(Compressed, List).

% Test cases
:- writeln('Testing compression:'),
   compress([1,2,4,5,6,7,8,10,15,16,17,20,21,22,23,24,25], CompressedList),
   writeln(CompressedList).

:- writeln('Testing decompression:'),
   decompress([1, 2, [4, 8], 10, [15, 17], [20, 25]], DecompressedList),
   writeln(DecompressedList).
```

Overview: Focuses on compressing a list of integers where continuous integers are encoded as ranges if the segment length exceeds two, otherwise kept unchanged.

Compression Process: Iterates through the list, encoding continuous integers as ranges ([start, end]) for long segments, preserving shorter segments as is.

Decompression Logic: Decompression expands each encoded range back to the continuous sequence of integers, accurately restoring the original list.

Exercise 3: Advanced Compression Techniques

Example:

compress([1,2,4,5,6,7,8,10,15,16,17,20,21,22,23,24,25], CompressedList).

decompress([1, 2, [4, 8], 10, [15, 17], [20, 25]], L).

Output:

```
Testing compression:
[1,2,[4,8],10,[15,17]]
Testing decompression:
[1,2,4,5,6,7,8,10,15,16,17,20,21,22,23,24,25]
true.

?- |
```

Summary and Conclusions

- We explored Prolog with these 3 exercises, and learned about its capabilities for logical reasoning, pattern matching, and effective recursion.

- These exercises demonstrate Prolog’s strength in handling complex list manipulations and data transformations.