Artificial Intelligence

Assignment 4

Claudia Schon

schon@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: 02.06.2022, 7:00 a.m.
Tutorial on: 02.06.2022 and 03.06.2022

GROUP HOLLERITH - SOLUTION

Group Members:

1. **Saborni Shernaj Binte Elahi (220202426) – (** saborni@uni-koblenz.de **)**
2. **M Rashedul Hasnat (220202415) – (**rhasnat@uni-koblenz.de**)**
3. **Basitur Rahman Chowdhury (218100976) – (**bchowdhury@uni-koblenz.de)
4. **Kamrun Nahar(220202410) – (** nahar@uni-koblenz.de**)**

1 Lists in Prolog: Common Elements (10 Points)

Develop a prolog predicate `common_element/2`, which tests if two lists have at least one element in common. The position of this common element does not have to be the same in the lists. Hints:

- The `/2` after the predicate name indicates the arity of the predicate to be developed.
- You are allowed to use predicate `member/2`.

For example the following queries should result in answer `yes`:

- `common_element([1,2,3],[a,1,b,c])`
- `common_element([1,2,3],[3,4,7])`
- `common_element([1,2,3],[3,4,1,7])`

For example following queries should result in answer `No`:

- `common_element([1,2,3],[a,b,c])`
- `common_element([1,2,3],[])`

SOLUTION:

(.pl file attached)

Code:

common_element(List1,List2):- member(X,List1),member(X,List2). %if X is in both list then it is a common elm

2 Lists in Prolog: Double Elements (10 Points)

Develop a prolog predicate `twice/2`, whose first argument is a list and whose second argument is the list to be obtained by writing each element of the list from the first argument twice.

Some example queries with their results:

- The query `twice([1,a,bla],X)` should result in the answer `X=[1,1,a,a,bla,bla]`.
- The query `twice([1,a,bla],[1,a,a,bla,bla])` should result in the answer `No`.
- The query `twice([1,a,bla],[1,1,a,a,bla,bla])` should result in the answer `Yes`.

SOLUTION:

(.pl file attached)

Code:

twice([], []). % rule for empty list

twice([H|T], [H,H|Rest]) :- twice(T, Rest). % recursive call for repeating items by putting head twice

3 Lists in Prolog: Compressing Lists (10 Points)

Develop a predicate `compress/2` which removes duplicates occurring in immediate succession in a list.

Some example queries with their results:

- The query `compress([1,2,2,3,4,4],X)` should return the response `X=[1,2,3,4]`.
- The query `compress([1,2,2,3,4,2,4],X)` should return the response `X=[1,2,3,4,2,4]`.
- The query `compress([a,b,c],X)` should return the response `X=[a,b,c]`.
- The query `compress([a,b,b,c],[a,b,c])` should return the response `Yes`.
- The query `compress([a,b,b,c],[a,b,b,c])` should return the response `No`.

SOLUTION:

(.pl file attached)

Code:

compress([],[]).

compress([H],[H]). %two base clauses. empty lists and one elemet lists are always compressed

compress([H,T|Ts],Lnew):-

H = T, %in case of H and T are equal i.e. repeated, the predicate will take only the second entry hence no repeatation.

compress([T|Ts],Lnew).

compress([H,T|Ts],[H|Lnew]):- %% when H is not repeated, it is put in the new list

H \=T, % T and H are not same and T is put inside the new list

compress([T|Ts],Lnew).

4 Lists in Prolog: Packing Lists (10 Points)

Write a prolog predicate `pack/2` which packs all identical elements occurring in immediate succession in a list into a sublist.

Some example queries with their results:

- The query `pack([1,2,2,2,3,4,4],X)` should return the response `X=[[1],[2,2,2],[3],[4,4]]`.
- The query `pack([1,2,3],X)` should return the response `X=[[1],[2],[3]]`.
- The query `pack([1,1,1],X)` should return the response `X=[[1,1,1]]`.
- The query `pack([1,2,2,2,3,4,4],[[1],[2,2,2],[3],[4,4]])` should return the response `Yes`.
- The query `pack([1,2,2,3,4,4],[1,2,3,4])` should return the response `No`.

SOLUTION:

(.pl file attached)

Code:

```
pack([], []).      % emptly list packing

pack([A], [[A]]).   % single list packing

pack([A,A|T], [[A|PH]|PT]):-
   pack([A|T], [PH|PT]).  % rule when two consecutive elms are same

pack([A,B|T], [[A]|PT]):-
   A \= B,
   pack([B|T], PT). % rule when different elms
```

5 Natural numbers (60 Points)

Predicates such as `sum` or `ordered` explicitly expect lists of numbers as arguments. As we have only defined first-order logic on arbitrary symbols, the question arises how numbers can be theoretically integrated into such a system. However, we can define a natural number to be either the constant 0 or a successor of a natural number. As a grammar, we may write:

$$n ::= 0 \mid \texttt{succ}(n)$$

That way, the number 1 can be encoded as `succ(0)`. This encoded can also be implemented in Prolog. The `add` predicate will then look as follows:

```
add(X,0,X).
add(X,succ(Y),succ(Z)) :- add(X,Y,Z).
```

Encode the following predicates without using any built in arithmetic operators this style:

a) `natNum(N)` is true if `N` is a natural number.

b) `isZero(N)` is true if `N` is zero.

c) `pred(N,P)` is true if `P` is the predecessor of `N`. Please note that the smallest number we can encode is 0—therefore, the predecessor of 0 is 0 again.

d) `eq(X,Y)` is true if `X` is equal to `Y`.

e) `minus(X,Y,Z)` is true if `Z` is the difference of `X` and `Y`.

f) `times(X,Y,Z)` is true f `Z` is the product of `X` and `Y`.

SOLUTION:

(.pl file attached)

Code:

```
% a) natNum(N) is true if N is a natural number.

%n = 0 | succ(n)

natNum(N) :- between(1, infinite, N).
```

```
% b) isZero(N) is true if N is zero.
isZero(N) :-
        sort([N,0],[_]).

% c) pred(N,P) is true if P is the predecessor of N. Please note that the smallest number
%we can encode is 0—therefore, the predecessor of 0 is 0 again.

pred(0,0).
pred(N,P) :- succ(P,N).

% d) eq(X,Y) is true if X is equal to Y
eq(X,Y) :- sort([X, Y], [_]).


% e) minus(X,Y,Z) is true if Z is the difference of X and Y.

minus(X, 0, X).
minus(X, Y, Z) :-
  succ(PredY, Y),
  succ(PredX, X),
  minus(PredX, PredY, Z).


% f) times(X,Y,Z) is true f Z is the product of X and Y.

times(0, _, 0).
times(X, Y, Z) :-
   X > 0,
   X1 is X - 1,
```

times(X1, Y, Z1),

Z is Y + Z1.