University of Malta
BSc(Hons.) IT Year II

Problem Solving Techniques in Artificial Intelligence

Lecture 3

Prolog Review

- Discussion of Exercise 1
- Recursion with Lists: list membership; list length; list append
- Declarative Arithmetic: successor notation.
- Maintaining Declarativity: extra conditions; extra clauses

Department of Computer Science and AI
1998/9

List Membership

Define m(X,L) which succeeds if X is a member of a list L.

```
m(X,[X|_]).

m(X,[_|Y]) :- m(X,Y).

?- m(c,[a,b,c]).

   Call:  (  7) m(c, [a, b, c]) ?

   Call:  (  8) m(c, [b, c]) ?

   Call:  (  9) m(c, [c]) ?

   Exit:  (  9) m(c, [c]) ?

   Exit:  (  8) m(c, [b, c]) ?

   Exit:  (  7) m(c, [a, b, c]) ?
```

Using m with an Unbound Element Variable

```
?- m(X,[a,b,c]).

X = a ;

X = b ;

X = c ;

No
```

There are three solutions and they are returned one by one.

Using m with an Unbound List Variable

```
?- m(a,X).

X = [a|_G345] ;

X = [_G344, a|_G348] ;

X = [_G344, _G347, a|_G351] ;

X = [_G344, _G347, _G350, a|_G354] ;

X = [_G344, _G347, _G350, _G353, a|_G357] ;
```

Note that in the second case the query has an infinite number of solutions which are returned one-by-one.

Length of a List

5. Define len(L,N) which binds N to the length (= number of elements in) list L.

```
len([ ],0).
len([_|Y],N) :- len(Y,N1),
                N is N1+1.
```

```
?- len([a,b],Z).
   Call:  (  7) len([a, b], _G304) ?
   Call:  (  8) len([b], _L142) ?
   Call:  (  9) len([], _L155) ?
   Exit:  (  9) len([], 0) ?
 ^ Call:  (  9) _L142 is 0+1 ?
 ^ Exit:  (  9) 1 is 0+1 ?
   Exit:  (  8) len([b], 1) ?
 ^ Call:  (  8) _G304 is 1+1 ?
 ^ Exit:  (  8) 2 is 1+1 ?
   Exit:  (  7) len([a, b], 2) ?

Z = 2
```

Representing Integers

6. Define conv(N,T) such that T is a representation for integer T

eg. conv(0,0) conv(4,s(s(s(s(0))))).

```
conv(0,0).
conv(N,s(C)) :-
    N1 is N-1,
    conv(N1,C).
?- conv(4,X).

X = s(s(s(s(0)))) ;
```

What's wrong with this? When we backtrack we enter the second clause with N=0. This causes an infinite loop. To fix this we must add another condition.

Modified Conv

```
conv(0,0).
conv(N,s(C)) :-
    N>0,           % <-- Here's the
    N1 is N-1,     % extra condition
    conv(N1,C).
```

```
?- conv(4,X).

X = s(s(s(s(0)))) ;
no
```

Now when we type the semicolon there are no more solutions.

Running Conv Backwards

Now we try to run conv with an unbound N so as to determine the numerical value of a given “successor represenation”.

```
?- conv1(N,s(s(0))).
[WARNING: Arguments are not
          sufficiently instantiated]
   Exception: (  9) _G199>0 ?
   Exception: (  8) conv1(_G199, s(s(0))) ?
[WARNING: Unhandled exception]
```

This doesn’t work because it enters clause 2 with N unbound. We can fix this by introducing a third clause to deal with this situation.

Modifying Conv to Run Backwards

```
conv2(0,0).
conv2(X,Y) :-        % var(X) succeeds only if
    var(X),          % its argument is an unbound
    unconv(Y,X).     % variable
conv2(N,s(C)) :-
    N>0,
    N1 is N-1,
    conv2(N1,C).
unconv(0,0).
unconv(s(X),N) :-
 unconv(X,N1),
 N1 is N+1.

?- conv2(N,s(s(s(0)))).
 N = 3 ;
[WARNING: Arguments are not sufficiently instantiated]
   Exception: (  9) _G219>0 ?
   Exception: (  8) conv2(_G219, s(s(s(0)))) ?
[WARNING: Unhandled exception]
```

Use of Cut

The problem is that after clause 3 is applying after clause 2. We must somehow deactivate clause 3 after clause 2. This is achieved with a cut.

```
conv2(0,0).
conv2(X,Y) :-
    var(X),
    !,                    % <----cut here
    unconv(Y,X).          % prevents the invocation
conv2(N,s(C)) :-          % of clause 3
    N>0,
    N1 is N-1,
    conv2(N1,C).

unconv(0,0).
unconv(s(X),N) :-
 unconv(X,N1),
 N1 is N+1.

?- conv2(N,s(s(s(0)))).
 N = 3 ;
 no
```

Arithmetic with Term Representation

Using the successor representation of numbers. define the sum1 relation so that sum(X,Y,N) gives all possible pairs X and Y whose sum is N

```
sum(0,X,X).

sum(s(X),Y,S) :-        % sum(s(X),Y,s(S)) :-
  sum(X,Y,S1),          %       sum(X,Y,S).
  S = s(S1).            % equivalent definition

?- sum(s(0),s(0),Z).
Z = s(s(0)) ;
No
?- sum(X,Y,s(s(0))).

X = 0
Y = s(s(0)) ;

X = s(0)
Y = s(0) ;

X = s(s(0))
Y = 0 ;
```

Concatenation of Lists

8. Define cat(L1,L2,L) such that L is the result of concatenating L1 and L2.

```
cat([],L,L).
cat([X|L1],L2,[X|L]) :- cat(L1,L2,L).

?- cat([a,b,c],[d,e,f],Z).
Z = [d, e, f, a, b, c] ;

?- cat(X,Y,[a,b,c])
X = []
Y = [a, b, c] ;

X = [a]
Y = [b, c] ;

X = [a, b]
Y = [c] ;

X = [a, b, c]
Y = [] ;
No
```

Summary

- A declaratively implemented relation ought to operate no matter what the configuration of unbound arguments.
- To achieve declarativity it is often necessary to insert extra clauses and/or conditions catering for a particular situation. When this happens, you typically don't want all the other clauses to apply. Use of the cut may be necessary.
- Often it is not actually necessary to achieve declarativity in this sense since you may know in advance that certain arguments will never actually be unbound. In such cases it is good practice to add a comment that specifies what the expectations of a particular predicate are.