Prolog Programming

Course: CS40002
Instructor: Dr. Pallab Dasgupta

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Basics

- The notion of instantiation

```
likes( harry, school )
likes( ron, broom )
likes( harry, X) :- likes( ron, X )
```

- Consider the following goals:

```
?- likes( harry, broom )
?- likes( harry, Y )
?- likes( Z, school )
?- likes( Z, Y )
```

Lists

- Lists can be written as:

```
            [ Item1, Item2, … ]
    or      [ Head | Tail ]
    or      [ Item1, Item2, … | Others ]
```

```
[a, b, c] = [a | [ b,c] ] = [a,b | [c] ] = [a,b,c | [ ] ]
```

- Items can be lists as well –

```
    [ [a,b], c, [d, [e,f] ] ]
```

Head of the above list is the list [a,b]

List examples

Membership:

```
member( X, [X, Tail] ).
member( X, [Head, Tail] ) :-
                        member( X, Tail ).
```

Concatenation:

```
conc( [ ], L, L ).
conc( [X | L1], L2, [X | L3] ) :-
                        conc( L1, L2, L3 ).
```

List examples

Adding in front:

```
add( X, L, [X | L] ).
```

Deletion:

```
del( X, [X | Tail], Tail ).
del( X, [Y | Tail], [Y | Tail1] ) :-
                                    del( X, Tail, Tail1).
```

List examples

Sublist:

```
sublist(S, L) :- conc(L1,L2,L), conc(S,L3,L2).
```

Permutation:

```
permutation( [ ], [ ] ).
permutation( [X | L], P ) :-
        permutation( L, L1 ), insert( X, L1, P ).
```

or

```
permutation( [ ], [ ] ).
permutation( L, [X | P] ) :-
        del( X, L, L1 ), permutation( L1, P ).
```

Arithmetic and Logical operators

- We have +, -, *, /, mod
  - The “is” operator forces evaluation
  - ?- X is 3/2. – will be answered by X=1.5
- We have
  - X > Y, X < Y, X >= Y, X =< Y
  - X =:= Y – X and Y are equal
  - X =\= Y – X and Y are not equal

Examples

- GCD of two numbers

```
gcd( X, X, X ).
gcd( X, Y, D ) :-
  X < Y, Y1 is Y – X, gcd( X, Y1, D ).
```

- Length of a list

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

Eight Queens Problem

```
solution( Queens ) :-
    permutation( [1,2,3,4,5,6,7,8], Queens ),
    safe( Queens ).

permutation( [ ], [ ] ).
permutation( [Head | Tail], Permlist ) :-
                permutation( Tail, PermTail ),
                del( Head, Permlist, PermTail ).
```

Eight Queens Problem (Contd.)

```
safe( [ ] ).
safe( [Queen | Others] ) :-
    safe( Others ), noattack( Queen, Others, 1 ).

noattack( _, [ ], _ ).
noattack( Y, [ Y1 | Ylist ], Xdist ) :-
    Y1 – Y =\= Xdist, Y – Y1 =\= Xdist,
    Dist1 is Xdist + 1, noattacks( Y, Ylist, Dist1 ).
```

Cuts – for controlling backtracking

```
C :- P, Q, R, !, S, T, U.
C :- V.
A :- B, C, D
?- A
```

- Backtracking within the goal list P, Q, R
- As soon as the cut is reached:
  - All alternatives of P, Q, R are suppressed.
  - The clause C:- V will also be discarded
  - Backtracking possible within S, T, U.
  - No effect within A :- B, C, D, that is, backtracking within B, C, D remains active.

Examples

- Finding the maximum of two numbers

  If X >= Y then Max = X, otherwise Max = Y.

```
max( X, Y, X ) :- X >= Y, !.
max( X, Y, Y ).
```

- Adding an element into a list without duplication

```
add( X, L, L ) :- member( X, L ), !.
add( X, L, [X | L] ).
```

Negation as failure

- Frodo likes all jewellery except rings

```
likes( frodo, X ) :-  ring( X ), !, fail.
likes( frodo, X ) :-  jewellery( X ).
```

- The “different” predicate:

```
different( X, X ) :- !, fail.
different( X, Y ).
```

Quicksort

```
quicksort( [ ], [ ] ).

quicksort( [ X | Tail ], sorted ) :-
   split( X, Tail, Small, Big ),
   quicksort( Small, SortedSmall ),
   quicksort( Big, SortedBig ),
   conc( SortedSmall, [ X | SortedBig ], Sorted ).
```

Quicksort

```
split( X, [ ], [ ], [ ] ).

split( X, [ Y | Tail ], [ Y | Small ], Big ) :-
        gt( X, Y ), !, split( X, Tail, Small, Big ).

split( X, [ Y | Tail ], Small, [ Y | Big ] ) :-
                    split( X, Tail, Small, Big ).
```