Lecture 12–Prolog examples:
Himalayan club, member, rem_duplicate, union, intersection
Introduction

- PROgramming in LOGic
- Emphasis on *what* rather than *how*
A Typical Prolog program

Compute_length([],0).
Compute_length([Head|Tail], Length):-
    Compute_length(Tail,Tail_length),
    Length is Tail_length+1.

High level explanation:
The length of a list is 1 plus the length of the tail of the list.

This is a declarative description of the computation.
## Facts

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>valuable(gold)</td>
<td>Gold is valuable.</td>
</tr>
<tr>
<td>owns(john,gold)</td>
<td>John owns gold.</td>
</tr>
<tr>
<td>father(john,mary)</td>
<td>John is the father of Mary</td>
</tr>
<tr>
<td>gives (john,book,mary)</td>
<td>John gives the book to Mary</td>
</tr>
</tbody>
</table>
Variables

- Always begin with a capital letter
  - `?- likes (john,X).
  - `?- likes (john, Something).
- But *not*
  - `?- likes (john,something)
Example of usage of variable

Facts:

likes(john,flowers).
likes(john,mary).
likes(paul,mary).

Question:

?- likes(john,X)

Answer:

X=flowers and wait
;
mary
;
no
Conjunctions

- Use ‘,’ and pronounce it as *and*.

- Example
  - Facts:
    - likes(mary, food).
    - likes(mary, tea).
    - likes(john, tea).
    - likes(john, mary)
  - ?-
    - likes(mary, X), likes(john, X).
    - Meaning *is anything liked by Mary also liked by John?*
Backtracking (an inherent property of prolog programming)

likes(mary,X), likes(john,X)

1. First goal succeeds. \( X=food \)
2. Satisfy \( \text{likes(john,food)} \)
Backtracking (continued)

Returning to a marked place and trying to resatisfy is called Backtracking

likes(mary,X), likes(john,X)

likes(mary,food)
likes(mary,tea)
likes(john,tea)
likes(john,mary)

1. Second goal fails
2. Return to marked place and try to resatisfy the first goal
Backtracking *(continued)*

1. First goal succeeds again, $X=tea$
2. Attempt to satisfy the $likes(john,tea)$
Backtracking (continued)

1. Second goal also succeeds
2. Prolog notifies success and waits for a reply
Rules

- **Statements about objects and their relationships**

- **Express**
  - *If-then conditions*
    - *I use an umbrella if there is a rain*
    - $use(i, \text{umbrella}) :- \text{occur(rain)}.$

- **Generalizations**
  - *All men are mortal*
  - $\text{mortal}(X) :- \text{man}(X).$

- **Definitions**
  - *An animal is a bird if it has feathers*
  - $\text{bird}(X) :- \text{animal}(X), \text{has\_feather}(X).$
Syntax

- `<head> :- <body>`
- Read ‘`:-`’ as ‘if’.
- E.G.
  - `likes(john,X) :- likes(X,cricket).`
  - “John likes X if X likes cricket”.
  - i.e., “John likes anyone who likes cricket”.
- Rules always end with ‘.’.
An example Prolog Program
Shows path with mode of conveyance from city \( C_1 \) to city \( C_2 \)

- `:-use_module(library(lists)).`
- `byCar(auckland,hamilton).`
- `byCar(hamilton,raglan).`
- `byCar(valmont,saarbruecken).`
- `byCar(valmont,metz).`
- `byTrain(metz,frankfurt).`
- `byTrain(saarbruecken,frankfurt).`
- `byTrain(metz,paris).`
- `byTrain(saarbruecken,paris).`
- `byPlane(frankfurt,bangkok).`
- `byPlane(frankfurt,singapore).`
- `byPlane(paris,losAngeles).`
- `byPlane(bangkok,auckland).`
- `byPlane(losAngeles,auckland).`
- `go(C1,C2) :- travel(C1,C2,L), show_path(L).`
- `travel(C1,C2,L) :- direct_path(C1,C2,L).`
- `travel(C1,C2,L) :- direct_path(C1,C3,L1), travel(C3,C2,L2), append(L1,L2,L).`
- `direct_path(C1,C2,[C1,C2,' by car']) :- byCar(C1,C2).`
- `direct_path(C1,C2,[C1,C2,' by train']) :- byTrain(C1,C2).`
- `direct_path(C1,C2,[C1,C2,' by plane']) :- byPlane(C1,C2).`
- `show_path([C1,C2,M|T]) :- write(C1), write(' to '), write(C2), write(M), nl, show_path(T).`
Prolog’s computation

- Depth First Search
  - Pursues a goal till the end
- Conditional AND; falsity of any goal prevents satisfaction of further clauses.
- Conditional OR; satisfaction of any goal prevents further clauses being evaluated.
What happens on failure

- REDO the immediately preceding goal.
Fundamental Principle of prolog programming

- Always place the more general rule \textit{AFTER} a specific rule.
CUT

- Cut tells the system that

*IF YOU HAVE COME THIS FAR*

*DO NOT BACKTRACK*

*EVEN IF YOU FAIL SUBSEQUENTLY.*

‘CUT’ WRITTEN AS ‘!’ ALWAYS SUCCEEDS.
Fail

- This predicate always fails.
- *Cut* and *Fail* combination is used to produce negation.
- Since the LHS of the neck cannot contain any operator, $A \Rightarrow \neg B$ is implemented as
  
  $$B :\ :- \ A, \ !, \ Fail.$$
Predicate Calculus

- Introduction through an example (Zohar Manna, 1974):
  - Problem: A, B and C belong to the Himalayan club. Every member in the club is either a mountain climber or a skier or both. A likes whatever B dislikes and dislikes whatever B likes. A likes rain and snow. No mountain climber likes rain. Every skier likes snow. *Is there a member who is a mountain climber and not a skier?*

- Given knowledge has:
  - Facts
  - Rules
A syntactically wrong prolog program!

1. member(a).
2. member(b).
3. member(c).
4. mc(X); sk(X) :- member(X) /* X is a mountain climber or skier or both if X is a member; operators NOT allowed in the head of a horn clause; hence wrong*/
5. like(X, snow) :- sk(X). /*all skiers like snow*/
6. \+like(X, rain) :- mc(X). /*no mountain climber likes rain; \+ is the not operator; negation by failure; wrong clause*/
7. \+like(a, X) :- like(b,X). /* a dislikes whatever b likes*/
8. like(a, X) :- \+like(b,X). /* a dislikes whatever b likes*/
9. like(a,rain).
10. like(a,snow).
?- member(X),mc(X),\+sk(X).
Correct (?) Prolog Program

member(a).
member(b).
member(c).
member(X):-\+mc(X),fail.
member(X).
member(X):-\+sk(X),!,fail.
member(X).
like(a,rain).
like(a,snow).
like(a,X) :- \+ like(b,X).
like(b,X) :- like(a,X),!,fail.
like(b,X).
mc(X):-like(X,rain),!,fail.
mc(X).
sk(X):- \+like(X,snow),!,fail.
sk(X).
g(X):-member(X),mc(X),\+sk(X),!.
Member (membership in a list)

member(X,[X|_]).
member(X,[_|L):- member(X,L).
Prolog’s way of making and breaking a list

Problem: to remove duplicates from a list

rem_dup([],[]).
rem_dup([H|T],L) :- member(H,T), !, rem_dup(T,L).
rem_dup([H|T],[H|L1]) :- rem_dup(T,L1).

Note: The cut ! in the second clause needed, since after succeeding at member(H,T), the 3rd clause should not be tried even if rem_dup(T,L) fails, which prolog will otherwise do.
Union (lists contain unique elements)

union([],Z,Z).
union([X|Y],Z,W):-
    member(X,Z),!,union(Y,Z,W).
union([X|Y],Z,[X|W]):- union(Y,Z,W).
Intersection  (lists contain unique elements)

intersection([],Z,[]).
intersection([X|Y],Z,[X|W]):-
    member(X,Z),!,intersection(Y,Z,W).
intersection([X|Y],Z,W):-
    intersection(Y,Z,W).