CS344: Introduction to Artificial Intelligence
(associated lab: CS386)

Pushpak Bhattacharyya
CSE Dept.,
IIT Bombay

Lecture 27, 28: Prolog
17th and 21st March, 2011
Introduction

- PROgramming in LOGic
- Emphasis on *what* rather than *how*

Diagram:
- Problem in Declarative Form
- Logic Machine
- Basic Machine
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, obtained by removing the first element of the list.

This is a declarative description of the computation.
Fundamentals

(absolutely basics for writing Prolog Programs)
Facts

- John likes Mary
  - like(john, mary)
- Names of relationship and objects must begin with a lower-case letter.
- Relationship is written first (typically the predicate of the sentence).
- Objects are written separated by commas and are enclosed by a pair of round brackets.
- The full stop character ‘.’ must come at the end of a fact.
# More 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>
Questions

- Questions based on facts
- Answered by matching

Two facts match if their predicates are same (spelt the same way) and the arguments each are same.

- If matched, prolog answers yes, else no.
- No does not mean falsity.
Prolog does *theorem proving*

- When a question is asked, prolog tries to match *transitively*.
- When no match is found, answer is *no*.
- This means *not provable* from the given facts.
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)

1. First goal succeeds. $X=\text{food}$
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, $X=\text{tea}$
2. Attempt to satisfy the $\text{likes(john,tea)}$
Backtracking (continued)

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

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

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, umbrella) :- occur(rain).
  - Generalizations
    - All men are mortal
    - mortal(X) :- man(X).
  - Definitions
    - An animal is a bird if it has feathers
    - bird(X) :- animal(X), 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 ‘.’.
Another Example

\textit{sister\textunderscore of} (X,Y):=\textit{ female} (X),

\textit{parents} (X, M, F),

\textit{parents} (Y, M, F).

\textit{X is a sister of Y is}

\textit{X is a female and}

\textit{X and Y have same parents}
Question Answering in presence of rules

- Facts
  - male (ram).
  - male (shyam).
  - female (sita).
  - female (gita).
  - parents (shyam, gita, ram).
  - parents (sita, gita, ram).
Question Answering: Y/N type: *is sita the sister of shyam?*

?- sister_of (sita, shyam)

- female(sita)
- parents(sita,M,F)
- parents(shyam,M,F)
- parents(sita,gita,ram)
- parents(shyam,gita,ram)

success
Question Answering: wh-type: whose sister is sita?

?- ?- sister_of (sita, X)

female(sita)

parents(sita,M,F)

parents(Y,M,F)

parents(sita,gita,ram)

parents(Y,gita,ram)

parents(shyam,gita,ram)

Success
Y=shyam
Rules

- Statements about *objects* and their *relationships*
- **Express**
  - *If-then conditions*
    - I use an umbrella if there is a rain
    - \( \text{use}(i, \text{umbrella}) :- \text{occur}(\text{rain}). \)
  - **Generalizations**
    - All men are mortal
    - \( \text{mortal}(X) :- \text{man}(X). \)
  - **Definitions**
    - An animal is a bird if it has feathers
    - \( \text{bird}(X) :- \text{animal}(X), \text{has_feather}(X). \)
Make and Break

Fundamental to Prolog
Prolog examples using making and breaking lists

%incrementing the elements of a list to produce another list
incr1([],[]).
incr1([H1|T1],[H2|T2]) :- H2 is H1+1, incr1(T1,T2).

%appending two lists; (append(L1,L2,L3) is a built-in function in Prolog)
append1([],L,L).
append1([H|L1],L2,[H|L3]) :- append1(L1,L2,L3).

% reverse of a list (reverse(L1,L2) is a built-in function in Prolog)
reverse1([],[]).
reverse1([H|T],L) :- reverse1(T,L1),append1(L1,[H],L).
Remove duplicates

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.
Member (membership in a list)

member(X,[X|_]).
member(X,[_|L]):- member(X,L).
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).
Prolog Programs are close to Natural Language

Important Prolog Predicate:

\[ \text{member}(e, L) /* true if } e \text{ is an element of list } L \]

\[ \text{member}(e,[e|L1]). /* e is member of any list which it starts } \]

\[ \text{member}(e,[_|L1]):- \text{member}(e,L1) /*otherwise } e \text{ is member of a list if the tail of the list contains } e \]

Contrast this with:

\[ \text{P.T.O.} \]
Prolog Programs are close to Natural Language, C programs are not

For (i=0; i<length(L); i++){
    if (e==a[i])
        break(); /*e found in a[]
}

If (i<length(L)){
    success(e,a); /*print location where e appears in a[]/*
else
    failure();
}

What is i doing here? Is it natural to our thinking?
Machine should ascend to the level of man

- A prolog program is an example of reduced man-machine gap, unlike a C program
- That said, a very large number of programs far outnumbering prolog programs gets written in C
- The demand of practicality many times incompatible with the elegance of ideality
- But the ideal should nevertheless be striven for
Prolog Program Flow, BackTracking and Cut

Controlling the program flow
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.
Control flow (top level)

Given

\[ g:~ a, b, c. \quad (1) \]
\[ g:~ d, e, f; g. \quad (2) \]

If prolog cannot satisfy (1), control will automatically fall through to (2).
Control Flow within a rule

Taking (1),

\[ g:- a, b, c. \]

If \( a \) succeeds, prolog will try to satisfy \( b \),
succeeding which \( c \) will be tried.

For ANDed clauses, control flows forward
till the ’.’, iff the current clause is \textit{true}.

For ORed clauses, control flows forward
till the ’.’, iff the current clause evaluates to \textit{false}.
What happens on failure

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

- Always place the more general rule 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, $A \rightarrow \sim B$ is implemented as

  $$B :- A, \!, \ Fail.$$
(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. belong(a).
2. belong(b).
3. belong(c).
4. mc(X); sk(X) :- belong(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).

?- belong(X),mc(X),+%sk(X).
Correct (?) Prolog Program

belong(a).
belong(b).
belong(c).
belong(X) :- 
+mc(X), 
+sk(X), !, fail.
belong(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) :- belong(X), mc(X), 
+ sk(X), !. /*without this cut, Prolog will look for next answer
on being given ‘;’ and return ‘c’ which is wrong*/
Himalayan club problem: working version

belong(a).
belong(b).
belong(c).

belong(X):-notmc(X),notsk(X),!, fail. /*contraposition to have horn clause
belong(X).

like(a,rain).
like(a,snow).
like(a,X) :- dislike(b,X).
like(b,X) :- like(a,X),!,fail.
like(b,X).

mc(X):-like(X,rain),!,fail.
mc(X).

notsk(X):- dislike(X,snow). /*contraposition to have horn clause
notmc(X):- mc(X),!,fail.
notmc(X).

dislike(P,Q):- like(P,Q),!,fail.
dislike(P,Q).

g(X):-belong(X),mc(X),notsk(X),!. 