Artificial Intelligence

Lecture 19

- **Prolog Programming for AI**

Prepared by:

Md. Mijanur Rahman, Prof. Dr.

Dept. of CSE, Jatiya Kabi Kazi Nazrul Islam University

Email: mijanjkkniu@gmail.com

Prolog Programming for AI

- **Outlines:**
  - Types of Objects
  - Operations on Lists
  - …

Types of Objects…

- Prolog provides for-
  - numbers,
  - atoms,
  - lists,
  - tuples, and
  - patterns.
- The types of objects that can be passed as arguments. Facts and rules are used as data and data is often passed in the arguments to the predicates.

Types of Objects…

- **Simple Types:**
  - **Numbers:** Include integer numbers and real numbers.
  - **Variables:** Variables are character strings beginning with a capital letter. For example: Result X.
  - **Atoms:** Atoms are either quoted character strings or unquoted strings beginning with a small letter; ‘Sarah Jones’ anna.

Types of Objects…

- **Composite Types:**
  - **Lists** are the most common data structure in Prolog. They are much like the array in that they are a sequential list of elements.
  - In addition to lists, Prolog permits arbitrary **patterns** as data. The **patterns** can be used to represent **tuples**.
  - Prolog does not provide an array type. But arrays may be represented as a list, and the multidimensional arrays as a **lists of lists**.

Types of Objects…

- **Lists:**
  - The list is a simple data structure widely used in non-numeric programming. A list is a sequence of any number of items. A list is designated in Prolog by square brackets ([ ]). An example of a list is:

    [dog, cat, mouse]
  - Elements in a Prolog list are ordered, even though there are no indexes.
  - The list is either empty or non-empty. The list can be viewd as consisting of two things:
    1) The first item, called the *head* of the list
    2) The remaining part of the list, called the *tail*.
  - Here, the **head** is dog and the **tail** is the list of [cat, mouse].

Types of Objects…

- **Lists:**
  - In general, the head can be anything and the tail has to be a list. The head and the tail are combined into a structure by a special functor:

    (Head, Tail)
  - So, the list can be represented as- (dog, .(cat, .(mouse, [])))
  - Also, we can list any number of elements by vertical bar '|':

    [Head | Tail] or [a, b, c] = [a | [b,c]] = [a,b|[c]]
  - The list can be represented by a special case of binary tree:

Types of Objects…

- **Tuples:**
  - Records or tuples are represented as patterns. Here is an example:

    book(title(lab_Manual), author(aaby, anthony), publisher(springer), date(1991))

  - The elements of a tuple are accessed by pattern matching.

    book(Title, Author, Publisher, Date).

    author(LastName, FirstName, MiddleName).

    publisher(Company, City).

Operations on Lists…

- Lists can be used to represent sets, although there is a difference:
  - The order of elements in a set does not matter while the order of items in list does;
  - Also the same element can occur repeatedly in a list.
- The most operations on lists are similar to those on sets. Among them are:
  - Checking whether some object is an element of a list (set membership)
  - Concatenation of two list, obtaining third list (union of sets)
  - Adding new item to a list, or deleting some object from it.

Operations on Lists…

- **Membership:** member(X, L)
  - Where X is an object and L is a list. The goal is true if X occurs in L. For example:
  - member(b, [a, b, c]) is true;
  - But, member(b, [a, [b, c]]) is not true.
  - The program for membership is based on the following-

    X is a member of L if either:

    1) X is the head of L, or
    2) X is a member of the tail of L.
  - This can be written in two clauses:

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

Operations on Lists…

- **Concatenation:** conc(L1, L2, L3)
  - Here, L1 and L2 are two lists, and L3 is their concatenation.
  - For example:

    ?- conc([a,b,c], [1,2,3], L).

    L = [a,b,c,1,2,3]
  - We can use ‘conc’ in the reverse direction for decomposing a given list into two lists:

    ?- conc(L1, L2, [a,b,c]).
    - L1 = [] L2 = [a,b,c]
    - L1 = [a] L2 = [b,c]
    - L1 = [a,b] L2 = [c]
    - L1 = [a,b,c] L2 = []

Operations on Lists…

- **Example-10. Union of two sets:**

  Prolog program:

```
union([], X, X) :- !.
union([X|R], Y, Z) :-
      member(X, Y), union (R, Y, Z).
union([X|R], Y, [X|Z]) :-
     union (R, Y, Z).
```

  Query:

```
?- union([a,b,c], [c,d,e], R).
R = [a,b,c,d,e]
```

- **Example-11. Intersection:**

  Prolog program:

```
intersect([], X, []) :- !.
intersect([X|R], Y, [X|T]) :-
   member(X, Y), intersect (R, Y, T).
intersect([X|R], Y, Z) :-
  intersect (R, Y, Z).
```

  Query:

```
?- intersect([a,b,c], [c,d,e], R).
R = [c]
```

Operations on Lists…

- **Example-12. Adding an Item:**

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

X is a new item added to the list L.
X becomes new head.

- In general, the operation of inserting X in any place in the list, can be defined by the clause:

```
insert(X, List, BiggerList) :-
        del(X, BiggerList, List).
```

- **Example-13. Deleting an Item:**

  del(X, L, L1)

- The list L1 is equal to the list L with the item X removed. The 'del' relation can be defined as follows:
  (1) If X is head then the list after deletion is the tail of the list.
  (2) If X is in tail then it is deleted from there.

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

Operations on Lists

- **Example-14. Sorting lists: Prolog BubbleSort program**

```
gt(X,Y) :- X > Y.
% A useful swap in List?
bsort(L, S) :-   swap(L, L1), !, bsort(L1, S).
% list is already sorted
bsort(S, S).
% Swap first two
swap([X,Y|R], [Y,X|R]) :- gt(X,Y).
Swap elements in tail
swap([Z|R], [Z|R1]) :-  swap(R, R1).
```

?- bsort([5,7,3,6,8,9,2,6], S).

S = [2, 3, 5, 6, 6, 7, 8, 9] ;

Prolog Programming for AI

TO BE CONTINUED…