AI LAB – 6

AIM: Study LIST data structure in PROLOG

Lists

So far we have only considered simple items as arguments to our programs. However in Prolog a very common data-structure is the list.

Lists themselves have the following syntax. They always start and end with square brackets, and each of the items they contain is separated by a comma. Here is a simple list
[a,freddie,A_Variable,apple]

Prolog also has a special facility to split the first part of the list (called the head) away from the rest of the list (known as the tail). We can place a special symbol | (pronounced 'bar') in the list to distinguish between the first item in the list and the remaining list.

For example, consider the following.

[first,second,third] = [A|B]
where *A = first* and *B=[second,third]*

The unification here succeeds. A is bound to the first item in the list, and B to the remaining list.

List Examples 1

Here are some example simple lists

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

[apple,pear,bananas,breadfruit]

[ ] /* this is a special list, it is called the empty list because it contains nothing */

Now lets consider some comparisons of lists:

[a,b,c] unifies with [Head|Tail] resulting in Head=a and Tail=[b,c]

[a] unifies with [H|T] resulting in H=a and T=[]

[a,b,c] unifies with [a|T] resulting in T=[b,c]

[a,b,c] doesn't unify with [b|T]

[] doesn't unify with [H|T]

[] unifies with []. Two empty lists always match

Answers to Lists Exercise

Do the following pairs of lists unify?

[a,d,z,c] and [H|T]

Yes *H* = a and *T* = [d,z,c]

[apple,pear,grape] and [A,pear|Rest]

Yes *A* = apple and *Rest* = [grape]

[a|Rest] and [a,b,c]

Yes *Rest* = [b,c]

[One] and [two|[]]

Yes *One* = two

[one] and [Two]

Yes *Two* = one

[a,b,X] and [a,b,c,d]

No *X* can not represent the two atoms c,d

List Searching

We can use lists within facts and rules. One common way of using lists is to store information within a list and then subsequently search for this information when we run our programs. In order to search a list, Prolog inspects the first item in a list and then goes on to repeat the same process on the rest of the list. This is done by using recursion. The search can either stop when we find a particular item at the start of the list or when we have searched the whole list, in which case the list to be searched will be the empty list. In order to do this, we have to be able to selectively pull the list apart. We have already seen how we can go about doing this. In the previous section we showed how to take the head and a tail of a list:

[Head|Tail]

This method constitutes the basis of the searching method. We shall use it to pull apart a list, looking at the first item each time, recursively looking at the tail, until we reach the empty list [], when we will stop.

List Searching: Example 1

Consider the following problem. How can I see if a particular item is on a particular list? For example I want to test to see if item apples is on the list [pears, tomatoes, apples, grapes]. One possible method of doing this is by going through the list, an item at a time, to see if we can find the item we are looking for. The way we do this in Prolog is to say that we could definitely prove an item was on a list if we knew that the target item was the first one on the list. ie.

```
on(Item,[Item|Rest]).      /* is the target item the head of the list */
```

Otherwise we could prove something was on a list if we could prove that although it didn't match the existing head of the list, it nonetheless would match another head of the list if we disregarded the first item and just considered the rest of the list i.e.

on(Item,[DisregardHead|Tail]):- on(Item,Tail).

We now have a program consisting of a fact and a rule for testing if something is on a rule. To recap, it sees if something is the first item in the list. If it is we succeed. If it is not, then we throw away the first item in the list and look at the rest.

Let's go through the last example again in more detail. Suppose we pose the query:

?- on(apples, [pears, tomatoes, apples, grapes]).

The first clause of on requires the first argument of on to match with the list head. However apples and pears do not match. Thus we must move on to the second clause. This splits the list up as follows:

DisregardHead = pears

Tail = [tomatoes,apples,grapes]

This now gives us the following goal in the body of our rule:

on(apples, [tomatoes, apples, grapes]).

Again, we see if apples and tomatoes match using our initial facts. Since they don't, we again use our second clause to strip off the current list head, giving us the new goal:

on(apples,[apples,grapes]). Since apples matches apples our first clause succeeds as does our query.

Exercises for YOU☺

1. Write a prolog program to check whether a number is a member of given list or not.
2. Write a prolog program to concatenate two lists giving third list.
3. Write a prolog program to find the last element in a given list.
4. Write a prolog program to reverse a list.
5. Write a prolog program to find the nth element of a list.
6. Write a prolog program to split a list in two lists such that one list contains negative numbers and one contains positive numbers.
7. Write a prolog program to delete first occurrence of a given element from a list.
8. Write a prolog program to delete every occurrences of a given element form a list.
9. Write a prolog program to generate sublist(S) of a given list.
10. Write a prolog program to check whether the given list is ordered or not.
11. Write a prolog program to add a given element at last of original list.
12. Write a prolog program to add a given element at start of original list.