Programming in Prolog

Part - 2

Loops

- Prolog does not support loops, so ‘for’, ‘while’, ‘do-while’ does not work in Prolog
- Still we can achieve same via *recursion*
- For example:

```
loop(0).

loop(N) :- N > 0, write(N), nl, S is N-1, loop(S).
```

- The above logic will simulate ‘for’ loop logic

Loops (contd..)

- Another way to achieve looping is via ***'repeat'*** built-in predicate
- For example:

```
correct_num(5).

correct_num(10).

predict_num(Input) :- write("Game: Guess a number"), nl, repeat, write("Your input?
"), nl,

                      read(Input), correct_num(Input), write("Correct!!"), nl.
```

Lists

- In Prolog, list is a common data-structure
- Example list:

  [apples, bananas, oranges, pears, mangos].

- With the help of " **|** " (bar) operator, Prolog splits any given list into two parts
  - Head: First element of the list (CAR: Contents of the Address part of the Register)
  - Tail: All remaining elements of list (CDR: Contents of the Decrement part of the Register; not the second element or last element)
- For example:

  [apples, bananas, oranges, pears, mangos] = [H|T]

  H = apples.

  T = [bananas, oranges, pears, mangos].

Lists (contd..)

- Prolog has built-in predicate *‘length/2’* which will calculate the length of a list
- Example:

  length([a, b, c], L). # L = 3

- The following code will simulate ‘*length/2’* built-in predicate

  size([], 0).   # size of empty list is 0

  size([_|T], N) :- size(T, N1), N is N1+1.   # increment N till Tail becomes NULL

Lists (contd)..

- Membership: Checking whether a given element is member of a list or not
- Logic:

```
is_member(X, [X|_]).

is_member(X, [_|T]) :- is_member(X, T).
```

- ‘X’ is a member of given list,
  - If X is head of the list
  - Or, Iterate through list till X becomes the head of the list
- If ‘X’ present then query will evaluated True otherwise False

Lists (contd)..

- Union: Combining two lists to make a new third list
- Logic:

```
list_append([], L, L).

list_append([H|T], L2, [H|R]) :- list_append(T, L2, R).
```

- First copy second list as Tail to new list
- One by one, add elements of first list to Head of new list in reverse order
- Prolog has '*append/3*' built-in predicate which will do the same

Lists (contd..)

- Reversing a list
- Logic:

    reverse_it([], []).

    reverse_it([H|T], L) :- reverse_it(T, L2), list_append(L2, [H], L).

- '*append/3*' will add elements of second list to Tail of new list
- To reverse a list, add each element of first list to second list one at a time, and then append the two lists to new list

Exercise

- Write a Prolog program to remove duplicates from a given list
- Write a Prolog program to perform intersection between two lists

Further Reading

- Controlling backtracking through '***cut (!)***' operator
- Negation As Failure (NAF)
- Usage of Accumulators

References

- https://discretemathisfun.wordpress.com/2009/12/07/looping-using-prolog/
- https://stackoverflow.com/questions/29857372/how-to-go-back-to-repeat-in-prolog
- https://www.javatpoint.com/looping-until-a-condition-is-satisfied
- https://www.doc.gold.ac.uk/~mas02gw/prolog_tutorial/prologpages/lists.html
- https://en.wikibooks.org/wiki/Prolog/Lists
- https://www.cs.toronto.edu/~hojjat/384w10/PrologTutorial2.pdf