Symbolic and Declarative Computing & AI

Coursework Project 3

BY

HASSAN AHMED

UOB# 14031239

APRIL 15, 2017

SUBMITTED TO

Dr: Junaid AKHTAR

Table of Contents

Task 1.

Description:

In that task I was supposed to find the length of the queue (which was given). For that I made a predicate named as “queue_length/1”. In that function I called a queue (which is list of items, which I have declared at top). After that I passed same queue (list) to auxiliary function “length_list/2”, which calls itself recursively to find out the length of the list by traversing through whole list and keep on adding 1 to the counter(variable) on each iteration until the whole traversal terminates. Then at the end it returns the length if the list in the counter (variable).

Code.

Main Function:

```
queue_length(L):-
        queue(P),
        length_list(P,L).
```

Auxiliary Function :

```
length_list([],0).
length_list([_|T],N):-
   length_list(T,N1),
   N is N1+1.
```

Output

```
1 ?- queue(Q).
Q = [adam, eve, tracy, pete, john, james, ruth, david].


2 ?- queue_length(Q).
Q = 8.
```

Task 2.

Description:

In that question I was supposed to write a query to remove head from the given list.

For that firstly I would initialize the list in the form of ([H|T]). Then I will call just head of the list in the queue. Then as a result I will get a list in which the head will have been removed. Which is desirous solution.

Code

```
remove_head:-
        queue([H|T]),
        retractall(queue([H|T])),
        assert(queue(T)).
```

Output:

```
1 ?- queue(Q).
Q = [adam, eve, tracy, pete, john, james, ruth, david].
2 ?- remove_head.
true.
3 ?- queue(Q).
Q = [eve, tracy, pete, john, james, ruth, david].
```

Task 3.

Description:

In that task I was supposed to remove the head of the given list. For that firstly I initialized the given queue (List) in the form of head and tail. For that I made a function "remove_head" in which I used built in predicate retractall (to remove the given queue (list)). Then I inserted just tail of the list in the queue by using the built in predicate assert. As a result I got a queue which contain just tail of the list, which was asked requirement.

Code :

```
remove_head:-
        queue([H|T]),
        retractall(queue([H|T])),
        assert(queue(T)).
```

Output:

```
1 ?- queue(Q).
Q = [adam, eve, tracy, pete, john, james, ruth, david].

2 ?- remove_head.
true.

3 ?- queue(Q).
Q = [eve, tracy, pete, john, james, ruth, david].
```

Task 4.

Description:

In that task I was supposed to remove the last element of the given queue (list). For that firstly I made an auxiliary function (helping function) named as “remove/2”, which take two arguments and traverse through whole list and calls itself recursively till the last element of the list. In each iteration it perseveres just the head of the list and return an empty list when one element remains in the list (which is last element). At the end it returns the list without last element, which is required solution.

After that I used that helping function in the main function named as “remove_last/0”. In that function firstly I called a queue which is already declared at the top. Then cleared the queue using the built in function retract all. Then I used built in function assert to insert the list, in which the last element has been removed (we will get that list from our auxiliary function).

<u>Code</u>

Auxiliary function:

```
remove([A],[]).
remove([H|T],[H|S]):-
```

```
        remove(T,S).
```

Main Function:

```
remove_last:-
        queue(Q),
        retractall(queue(Q)),

        remove(Q,A),
        assert(queue(A)).
```

Output:

```
1 ?- queue(A).
A = [adam, eve, tracy, pete, john, james, ruth, david].

2 ?- remove_last.
true .

3 ?- queue(A).
A = [adam, eve, tracy, pete, john, james, ruth].
```

It is cleared from the output that after calling the remove_last function the last element has been removed from the list.

Task 5.

Description:

In that function I was asked to write the function which will take on element as an argument and that element will be paced at the start of the list. For that I made a function named as "join_front/1" which takes just one element as an argument which will be placed at the start of the list. In that function firstly I called the queue (Q), which is already declared at the top. Then I used the built in function append to get desirous result. The append function takes three parameter as an argument.

First the given input (which we will place at the start), second the given list (queue) and third argument in which I will keep my result.

After that I used built in function retract all to clear the list then insert the updated list (which is third argument of the append function) in the queue using built in function assert. As a result I got a list in which the given input has been appended at the start of the queue.

<u>Code</u>

```
join_front(S):-
        queue(T),
        append([S],T,Z),
        retractall(queue(T)),
        assert(queue(Z)).
```

Output:

```
4 ?- queue(A).
A = [adam, eve, tracy, pete, john, james, ruth].

5 ?- join_front(hassan).
true.

6 ?- queue(A).
A = [hassan, adam, eve, tracy, pete, john, james, ruth].
```

You can see that after calling the join_front function, the argument (hassan) has been appended at start.

Task 6.

Description:

In that task I was asked to append the given input at the end of the queue (List). That question is quite opposite of question 5. For that task, I made a function named as "join_last/1", which takes one argument as an input and then append that input at the end of the queue. In that function firstly I called the queue (Q), which is already declared at the top. Then I used the built in function append to get desirous result. The append function takes three parameter as an argument.

First the given input (which we will place at the end of the queue), second the given list (queue) and third argument in which I will keep my result.

After that I cleared the queue using built in function retract all then insert the list (which is third argument of the append function) in the queue using built in function assert. That list will be modified list in which the input parameter has been placed at the end of the list.

Code

```
join_Last(L):-
        queue(N),
        append(N,[L],Z),
        retractall(queue(N)),
        assert(queue(Z)).
```

Output:

```
1 ?- queue(A).
A = [adam, eve, tracy, pete, john, james, ruth, david].

2 ?- join_Last(ahmed).
true.

3 ?- queue(A).
A = [adam, eve, tracy, pete, john, james, ruth, david, ahmed].
```

You can see that after applying the join_last/1 function on the queue, the given input has been appended at the end of the list, which is our required solution.

Task 7.

Description.

In that question I was asked to make a function to shift the head of the list from start to end of the list. In the solution of that question I again used the built in function append in order to get the desirous results.

Firstly I made an auxiliary function named as “send_back/2”, which take two arguments which are the given list and one for storing results. In that function I used append function to append the head (first element) of the list, at the end of the tail. Then I stored the resulted list in the in third argument of the append function (Which is N in my code).

Then I used that auxiliary function in the main function, named as” send_to_back/0”. In that function firstly in initialized the queue in the form of head and tail (H|T). Then I called the auxiliary function “send/back/2” and pass the relevant arguments. Then I used the built in function retract all and assert to get required results.

At the end I got a list in which the head of the list has been appended and the end of the list.

<u>Code.</u>

Auxiliary function:

```
send_back([H|T],N):-
        append(T,[H],N).
```

Main Function:

```
send_to_back:-
        queue([H|T]),
        send_back(([H|T]),X),


        retractall(queue([H|T])),
        assert(queue(X)).
```

In the X there will be required list.

Output:

```
4 ?- queue(A).
A = [adam, eve, tracy, pete, john, james, ruth, david, ahmed].
```

```
5 ?- send_to_back.
true.


6 ?- queue(A).
A = [eve, tracy, pete, john, james, ruth, david, ahmed, adam].
```

You can see that the adam(head of the list) has been moved at the end of the list.

Task 8.

Description:

 In that task I was supposed to make a function which will traverse through whole list of elements and will remove every second element from the list.

For that I made an auxiliary function "remove_s/2", which takes two parameters as argument. For that function I made two base cases (one for even numbers of the element in the list and one for odd numbers of elements in the list).

(In short that function preserve the head of list and removes the second head of the list and then update the list. Same is repeated till the end of the list).

Then that function traverse through whole list and calls itself recursively till the end of list in order to get the desirous results.

Then I made main function named as remove_every in which I used auxiliary function to get desirous results. Then with the help of built in functions retract all firstly I cleared whole queue, then by using assert function in inserted the updated list in the queue.

<u>Code.</u>

Auxiliary Function

```
remove_s([],[]).

remove_s([J],[J]).

remove_s([H,_|T],[H|W]):-

        remove_s(T,W).
```

Mian Function:

```
remove_every:-

        queue([H|T]),
        remove_s(([H|T]),D),


        retractall(queue([H|T])),
        assert(queue(D)).
```

Output:

```
1 ?- queue(A).
A = [adam, eve, tracy, pete, john, james, ruth, david].


2 ?- remove_every.
true.


3 ?- queue(A).
A = [adam, tracy, john, ruth].
```

You can see from output that the every second element has been removed from the given queue. Which is our required solution.

Task 9.

Description:

In that question I was asked to make the function which can find the behind element of given input element. For example the given list is [a,b,c] and the given input is “a”, and we are supposed to return the “b”.

Foe that I made an auxiliary function named as “back” which take three arguments. One the input element, second the given list (in which we have to find) and in third parameter we will return the result. That function calls itself recursively to find the desirous result. During each recursive call when it match the base case, it returns the result. While the base case is satisfied when the given input unify with element (same element as given in input).

After that I made main function named as “behind_back/2” which takes two elements, one the input element and in second we will return the result. In that function firstly I initialized the queue. Then in called the auxiliary function and passed the relevant arguments to it. Then I used the built in function retractall to clear the queue. Then by using the built in function assert I inserted the resulted element in the queue.

Code.

Auxiliary Function:

```
back(A,[A,B|_],B).
back(X,[_|T],B):-
        back(X,T,B).
```

Main Function:

```
behind_back(X,B):-
        queue([H|T]),
        back(X,[H|T],B),
        retractall(queue([H|T])),
        assert(queue(B)).
```

Output:

```
1 ?- queue(A).
A = [adam, eve, tracy, pete, john, james, ruth, david].

2 ?- behind_back(adam,B).
B = eve
```

You can see from the output that when I gave the input “adam” from the list it returned “eve”, which is next element of the adam.