CS 560: Homework 5(a)

Eric Stevens

November 4, 2018

Question :

Question 1:

Reflexive, Symmetric, and Transitive:

- X = X
- X = Y ← Y = X
- X = Z ← X = Y ^ Y = Z

Axiom Schema:

- planet(X) ← planet(Y) ^ X = Y

Question 2:

?planet(morningstar)

yes ← planet(morningstar)

- **Use:** planet(X) ← planet(Y) ^ X = Y
- **Sub:** {X/morningstar}

yes ← planet(Y) ^ morningstar = Y

- **Use:** planet(venus)
- **Sub:** {Y/venus}

yes ← morningstar = venus

- **Use:** morningstar = venus
- **Sub:** {}

yes ← true.

Question 3:

- eveningstar = venus (treated as morningstar => venus)

- morningstar = venus (treated as morningstar => venus)

Question 4:

?planet(morningstar)

yes ← planet(morningstar)

- **Paramodulation with:** morningstar => venus

yes ← planet(venus)

- **Use:** planet(venus)
- **Sub:** {}

yes ← true.

Question 5:

It is said in the assignment that we should build a predicate that ensures the entry of new nodes into the clique list are greater than nodes already in the clique list. The way the KB is designed, in all `c(a,b)` `a < b` and thus KBs structure can be taken advantage of to impose the desired behavior without the need for a clause to enforce it. Therefore this code does not have the extra clause and takes advantage of the structure of KB in order to acomplish the same goal.

clique_Q5.pl

```
/*Makes sure c(Node,Element) exists for every element in list*/
connected(Node, [X]) :-
        c(Node,X).
connected(Node,[First|Rest]) :-
        c(Node,First),
        connected(Node,Rest).

/*Ensures that Node does not exist in list*/
not_member(Node,[X]) :-
        not(Node=X).
not_member(Node,[Top|Rest]) :-
        not(Node=Top),
        not_member(Node,Rest).

/*Builds clique of size N from knowledge base as list*/
clique(1,[X]) :-
        node(X).
clique(N, [Top|List]) :-
        N >1,
        Nminus is N-1,
```

```
clique(Nminus,List),
node(Top),
not_member(Top,List),
connected(Top,List).
```

Question 6:

assert, retract, and retractall:

`assert` and `retract` are used to add and remove clauses from the knowledge base respectivly. `assert` adds the atom or rule to the bottom of the knowldge base. `retract` removes the first fact or atom it can unify with from the knowledge base. `retractall` will remove anything it can unify with from the knowledge base. When recursing through a function, a view of the knowledge base state at the time of the predicate call is maintained in what is refered to as '*logical update view*'. Therefore, when backtracking, even if something has been retracted later in the recursion, operations occouring higher up in the backtrack will operate on the view of the knowledge base when it was origionally called.

Our Usage:

In the code used to count the number of times `getnode` is called, there is a simple scheme to add and remove grounded `cnt` clauses to and from the knowledge base. When `countclique` is called it resets the `cnt` to `0` by first eliminating `cnt` fact from the knowledge base using `retractall(cnt(_))`. Now that the knowledge base is free of any `cnt` facts, `assert(cnt(0))` is called to push the fact `cnt(0).` into the knowledge base. Now the `clique` funtion can be called. In `clique` the call to `node` has been replaced with `getnode`. `getnode` acomplishes the goal of `node`, ensuring that a node exists. If the node exists it moves on to `retract(cnt(Cnt))` which simultaniously removes the `cnt` fact from the knowledge base and unifies `Cnt` with whatever value was held in the `cnt` fact that was removed. `Cnt` is incremented to `Cnt1` and then `assert(cnt(Cnt1))` is used to update the knowledge base with the incremented count. This occours each time the `getnode` predicate is called.

Finally, there is a second call to `countclique`. This predicate runs after the completion of the above described version. Its form `countclique(_,_)` means that it will accept any two terms, and therefore will always run after the first `countclique`. Its line `cnt(N)` simpily unifies `N` with the first `cnt` fact it comes to in the knowledge base, which at this point should be the last `assert` called in `getnode`. `N` should be the number of times that `getnode` was called and can now be printed to show how many nodes were examined in the process of finding the k-clique.

Question 7:

1-Clique:

- getnode calls: 15
- solutions: 15

2-Clique:

- getnode calls: 240
- solutions: 23

3-Clique:

- getnode calls: 585
- solutions: 13

4-Clique:

- getnode calls: 780
- solutions: 5

5-Clique:

- getnode calls: 855
- solutions: 1

Code and results below.

clique_Q7.pl

```
/*Makes sure c(Node,Element) exists for every element in list*/
connected(Node, [X]) :-
        c(Node,X).
connected(Node,[First|Rest]) :-
        c(Node,First),
        connected(Node,Rest).

/*Ensures that Node does not exist in list*/
not_member(Node,[X]) :-
        not(Node=X).
not_member(Node,[Top|Rest]) :-
        not(Node=Top),
        not_member(Node,Rest).

/*Builds clique of size N from knowledge base as list*/
clique(1,[X]) :-
        getnode(X).
clique(N, [Top|List]) :-
        N >1,
        Nminus is N-1,
        clique(Nminus,List),
        getnode(Top),
        not_member(Top,List),
        connected(Top,List).
```

Results

```
?- countclique(1,X).
X = [1] ; X = [2] ; X = [3] ; X = [4] ; X = [5] ;
X = [6] ; X = [7] ; X = [8] ; X = [9] ; X = [10] ;
X = [11] ; X = [12] ; X = [13] ; X = [14] ; X = [15] ;
15
true.

?- countclique(2,X).
X = [1, 2] ; X = [1, 3] ; X = [2, 4] ; X = [1, 5] ;
X = [3, 5] ; X = [4, 6] ; X = [1, 7] ; X = [3, 7] ;
X = [5, 7] ; X = [1, 8] ; X = [2, 8] ; X = [6, 8] ;
X = [1, 11] ; X = [3, 11] ; X = [5, 11] ; X = [7, 11] ;
X = [8, 11] ; X = [2, 12] ; X = [9, 12] ; X = [10, 12] ;
X = [13, 14] ; X = [13, 15] ; X = [14, 15] ;
240
true.

?- countclique(3,X).
X = [1, 3, 5] ; X = [1, 3, 7] ; X = [1, 5, 7] ;
X = [3, 5, 7] ; X = [1, 2, 8] ; X = [1, 3, 11] ;
X = [1, 5, 11] ; X = [3, 5, 11] ; X = [1, 7, 11] ;
X = [3, 7, 11] ; X = [5, 7, 11] ; X = [1, 8, 11] ;
X = [13, 14, 15] ;
585
true.

?- countclique(4,X).
X = [1, 3, 5, 7] ;
X = [1, 3, 5, 11] ;
X = [1, 3, 7, 11] ;
X = [1, 5, 7, 11] ;
X = [3, 5, 7, 11] ;
780
true.

?- countclique(5,X).
X = [1, 3, 5, 7, 11] ;
855
true.
```