Twitbook

Philip Lassen (vgh804), Nicolas Larsen (vgn209

December 2018

1 Implementation

(a) **likes(G, X, Y)** : We use a helper predicate **mem**, which check takes a person and a list, and check if the person is in the list. So to implement likes we simply go through G and once we find X, make sure that calling mem with Y and X's friends list is true.

(b) **dislikes(G, X, Y)** : X dislikes Y only when all three of the following things are true: Y likes X, X is not Y, and Y is not in X's friendlist. Thus it makes us to call three predicates for each of these facts. Thus we call **likes(G, Y, X)**, **different(G, X, Y)** and **notLikedBy(G, G, X, Y)**. notLikedBy checks if Y is not in X's friend list.

(c) **popular(G, X)**: In order to check if everybody that X likes, likes them back we need to keep track of the Graph throughout. So we call a predicate **pop1(G, X, G)** to keep the state. pop1 Gets X's friend list while keeping the graph and then calls another predicate **allLike(G, X, Friends)**. This predicate goes though Friends and calls **likes(G, Y, X)** for all Y in Friends.

(d) **outcast(G, X)**: outcast is implemented in exactly the same way as **popular(G, X)**. In order to check if everybody that X likes, dislikes them back we follow the same strategy as in popular. So we call a predicate **out1(G, X, G)** to keep the graph. out1 Gets X's friend list while keeping the graph and then calls another predicate **allDisLike(G, X, Friends)**. This predicate goes through Friends and calls **disLikes(G, Y, X)** for all Y in Friends.

(e) **friendly(G, X)**: Again we have to keep track of the Graph throughout so we call a helper predicate **keepGraph(G, X, G)**. In keepGraph we go through every member of G and get their friends list and call a predicate **chill(G, X, Y)**. chill is true when Y does not like X or when Y likes X and X likes Y.

(f) **hostile(G, X)**: We follow the same strategy as friendly above. The difference essentially being the **unchill(G, X, Y)** predicate. This predicate

evaluates to true when Y likes X and X does not like Y or when Y does not like X.

(g) **admires(G, X, Y)** : If the Graph was acyclic this would be very ease to implement but since there are cycles we can end up in an infinite loop so we must be careful. Thus we have to keep track of a list of nodes/friends that we have already visited. Thus we make a helper predicate **admires(G, X, Y, V)**. The V is a list which we can use to keep track of Nodes we have visitited. We then call the predicates different, likes, notInList and admires recursively. The predicate stops once we have visited all the nodes in the cycle which is how we terminate.

2 Testing

We first started by checking to see how we did we the online TA. We were able to pass all the tests for the predicates we implemented. We were only missing two predicates which were the indifferent and `same_world` predicates. When it came to our own testing we created both cyclical and acyclic graphs with conneceted and nonconnected components to make sure that our code worked for different types of graphs. We did this in the prolog REPL. This is because both of us were using GNU-prolog, which does not offer the same testing suites as swi-prolog.

3 Assessment

We Implemented twit-book without considering computational complexity. It is possible that our predicates perform poorly for this reason, however this is not something we analyzed. We were also not able to implement Same World. Part of the reason for this is that we found it difficult to make lists in prolog. Making lists in prolog is quite different then other languages. We looked at the example for append in order to understand how to create a list and even with the solution it took us a long time to understand.

Overall I think we did a good job of implementing the prolog predicate's. The only issues we ran into was when the predicates required generating lists. This is something that is very necessary for many graph algorithms so it is something we do need to improve on. But we ended up passing all the tests for the predicates we implemented, which were all but two.

4 Code

```
mem(Y, [Y| _]).
mem(Y, [ _ | Tail]) :- mem(Y, Tail).
likes([person(X, Friends)|_], X, Y) :- mem(Y, Friends).
likes([_ | Tail], X, Y) :- likes(Tail, X, Y).
```

```
/**
* X likes Y, Y no likes X, Y in X
**/

different([ _ | Tail], X, Y) :- different(Tail, X, Y).
different([person(X, _) | Tail], X, Y) :- mem(person(Y, _), Tail).
different([person(Y, _) | Tail], X, Y) :- mem(person(X, _), Tail).

dislikes(G, X, Y) :- likes(G, Y, X), different(G, X, Y), notLikedBy(G, G, X, Y).

notLikedBy(G, [person(X, Friends) | _], X, Y) :- notInList(G, Friends, Y).
notLikedBy(G, [ _ | Tail], X, Y) :- notLikedBy(G, Tail, X, Y).
notLikedBy(G, [], X, _) :- different(G, X, X).

notInList(G, [Z | Tail], Y) :- notInList(G, Tail,  Y), different(G, Z, Y).
notInList(_, [], _).

popular(G, X) :- pop1(G, X, G).
pop1([person(X, Friends)|_], X, G):- allLike(G, X, Friends).
pop1([_ | Tail], X, G) :- pop1(Tail, X, G).
allLike(G, X, [Head | Tail]) :- likes(G, Head, X), allLike(G, X, Tail).
allLike(_, _, []).

outcast(G, X) :- out1(G, X, G).
out1([person(X, Friends)|_], X, G) :- allDisLike(G, X, Friends).
out1([_ | Tail], X, G) :- out1(Tail, X, G).
allDisLike(G, X, [Head | Tail]) :- dislikes(G, Head, X), allDisLike(G, X, Tail).
allDisLike(_, _, []).


friendly(G, X) :- keepGraph(G, X, G), isMem(G, X).
keepGraph([person(Y, _) | Tail], X, G) :- chill(G, X, Y), keepGraph(Tail, X, G).

keepGraph([], _, _).
chill(G, X, Y) :- likes(G, X, Y), likes(G, Y, X).
chill(G, X, Y) :- notLikedBy(G, G, Y, X).

hostile(G, X) :- keepGraphU(G, X, G), isMem(G, X).
keepGraphU([person(Y, _) | Tail], X, G) :- unchill(G, X, Y),
keepGraphU(Tail, X, G).
keepGraphU([], _, _).
unchill(G, X, Y) :- dislikes(G, X, Y), likes(G, Y, X).
unchill(G, X, Y) :- notLikedBy(G, G, Y, X).
```

```
admires(G, X, Y) :- admires(G, X, Y, [X]).
admires(G, X, Y, V) :- different(G, X, Y),
likes(G, X, N), notInList(G, V, N), admires(G, N, Y, [N | V]).
admires(G, X, Y, _) :- likes(G, X, Y).


indifferent(G, X, Y) :- different(G, X, Y), list(G, X, V), tracker(G, V, Y, [X]), notLik


isMem([person(X, _) | _], X).
isMem([ _ | Tail], X) :- isMem(Tail, X).

list([person(X, Friends)|_], X, Friends).
list([_ | Tail], X, Friends) :- list(Tail, X, Friends).
```