Haskell intro

Assignment 2

Kai Arne S. Myklebust, Silvan Adrian

Handed in: October 3, 2018

Contents

1 Design/Implementation

We implemented all needed predicates by not using any built-in Prolog predicates or control operators, which we were restricted to use. So we had to implement predicates for select and not ourselves. In addition we defined other helper predicates to solve the tasks.

2 Code Assessment

We wrote Tests for testing all the defined predicates, which test either negative or positive responses of them. Also we used 2 different Lists to be able to test the predicates on 2 Lists.

OnlineTA also returned no errors by testing out our predicates.

We needed to implement a lot of new helper predicates. We tried to give them good names and place them after they were used, but sometimes it could get a little bit confusing what does what and where everything is defined.

A Code Listing

```
/*
    Assignment 3

    Authors: Kai Arne S. Myklebust, Silvan Adrian

*/

g1([person(kara, [barry, clark]),
 person(bruce, [clark, oliver]),
 person(barry, [kara, oliver]),
 person(clark, [oliver, kara]),
 person(oliver, [kara])]).

g2([person(batman, [green_arrow, superman]),
    person(green_arrow, [supergirl]),
    person(supergirl, [flash, superman]),
    person(flash, [green_arrow, supergirl]),
    person(superman, [green_arrow, supergirl])]).

a([p(kara,supergirl),
    p(bruce,batman),
    p(barry,flash),
    p(clark,superman),
    p(oliver,green_arrow)]).

% Helpers

/* Checks if an elem is member of list */
isMember(Head,[Head|_]).
isMember(Head,[_|Tail]) :-
    isMember(Head,Tail).

/* different succeeds whenever X and Y are different members of the r
different(G, X, Y) :-
    selectList(person(X, _), G, G2),
    selectList(person(Y, _), G2, _).

/* extracts X from a list */
selectList(X, [X|Tail], Tail).
```

```
selectList(Elem, [Head|Tail], [Head|Rest]) :-
    selectList(Elem, Tail, Rest).

/* Gets the friends list from a person */
getFriends([person(X, Friends)|_], X, Friends).
getFriends([_|T], X, Friends) :- getFriends(T, X, Friends).

% Task a
likes(G,X,Y) :-
    getFriends(G, X, Friends),
    isMember(Y, Friends).

% Task b
dislikes(G, X, Y) :-
    different(G, X, Y),
    likes(G, Y, X),
    notLikes(G, X, Y).

notLikes(G, X, Y) :-
    getFriends(G, X, Xfriends),
    isNotFriend(G, Y, Xfriends).

isNotFriend(_, _, []).
isNotFriend(G, X, [Friend|RestFriends]) :-
    different(G, X, Friend),
    isNotFriend(G, X, RestFriends).

equal(X,X).

% Helpers end

% Level 1
% Task c
popular(G, X) :-
    getFriends(G, X, Friends),
    allLikingX(G, X, Friends).

allLikingX(_, _, []).
allLikingX(G, X, [Head | Tail]) :-
    likes(G, Head, X),
    allLikingX(G, X, Tail).
```

```
% Task d
outcast(G, X) :-
    getFriends(G, X, Friends),
    allDislikeX(G, X, Friends).

allDislikeX(_, _, []).
allDislikeX(G, X, [Head | Tail]) :-
    dislikes(G, Head, X),
    allDislikeX(G, X, Tail).

% Task e
friendly(G, X) :-
    isMember(person(X, _), G),
    checkFriendliness(G, G, X).

checkFriendliness([], _, _).
% skip where name is same as X
checkFriendliness([person(Name, _)|Tail], G, Name) :-
    checkFriendliness(Tail, G, Name).
% check for like in both directions
checkFriendliness([person(Name, _)|Tail], G, X) :-
    likes(G, Name, X),
    likes(G, X, Name),
    checkFriendliness(Tail, G, X).
% check that there is no friend(likes) relation
checkFriendliness([person(_, Friends)|Tail], G, X) :-
    isNotFriend(G, X, Friends),
    checkFriendliness(Tail, G, X).

% Task f
hostile(G, X) :-
    isMember(person(X, _), G),
    checkHostileness(G, G, X).

checkHostileness([], _, _).
% skip where name is same as X
checkHostileness([person(Name, _)|Tail], G, Name) :-
    checkHostileness(Tail, G, Name).
% check for like in one direction and dislike in the other
checkHostileness([person(Name, _)|Tail], G, X) :-
```

```
    likes(G, Name, X),
    dislikes(G, X, Name),
    checkHostileness(Tail, G, X).
% check that there is no friend(likes) relation
checkHostileness([person(_, Friends)|Tail], G, X) :-
    isNotFriend(G, X, Friends),
    checkHostileness(Tail, G, X).

% Level 2
% Task g
admires(G, X, Y) :-
    different(G, X, Y),
    checkAdmires(G, X, Y, [X, Y]).

% check first for direct like, else check like-chain with list of all
checkAdmires(G, X, Y, _) :- likes(G, X, Y).
checkAdmires(G, X, Y, List) :-
    likes(G, X, Z),
    isNotFriend(G, Z, List), % checks that there is no loop
    checkAdmires(G, Z, Y, [Z|List]).

% Task h
indifferent(G, X, Y) :-
    different(G, X, Y),
    getAdmirers(G, [], [X], Admirers),
    isNotFriend(G, Y, Admirers).

getAdmirers(_, AdmiresList, [], AdmiresList).
getAdmirers(Original, AdmiresList, [ToCheck|ToBeChecked], L) :-
        getFriends(Original, ToCheck, ToCheckFriends),
        appendlist(AdmiresList, ToBeChecked, CurrentAdmires),
        rmSublist(Original, CurrentAdmires, ToCheckFriends, NewFriend
        appendlist(ToBeChecked, NewFriendsToBeChecked, NewToBeChecked
        appendlist(AdmiresList, [ToCheck], NewCheckedFriends),
        getAdmirers(Original, NewCheckedFriends, NewToBeChecked, L).

appendlist([], X, X).
appendlist([T|H], X, [T|L]) :-
    appendlist(H, X, L).

rmSublist(_, _, [], []).
```

```
rmSublist(Original, Y, [X|W], Z):-
    isMember(X, Y),
    rmSublist(Original, Y, W, Z).
rmSublist(Original, Y, [X|W], [X|Z]) :-
    isNotFriend(Original, X, Y),
    rmSublist(Original, Y, W, Z).

% Level 3
% Task i
same_world(G, H, A) :-
    sameNumberOfNames(G, H),
    getNames(G, Gnames),
    getNames(H, Hnames),
    genA(Gnames, Hnames, A),
    testA(G, H, A).

% generates the A list
genA([], _, []).
genA([GHead|GTail], Hnames, [(GHead,N)|A]) :-
    selectList(N, Hnames, HnamesNew),
    genA(GTail, HnamesNew, A).

% checks that the A list works together with G and H
testA([], _, _).
testA([GHead|GTail], H, A) :-
    equal(GHead, person(Name1, Friends1)),
    isMember((Name1, Name2), A),
    getFriends(H, Name2, Friends2),
    sameNumberOfNames(Friends1, Friends2),
    sameFriends(Friends1, Friends2, A),
    testA(GTail, H, A).

sameFriends([], _, _).
sameFriends([Name1|Tail], Friends2, A) :-
    isMember((Name1, Name2), A),
    isMember(Name2, Friends2),
    sameFriends(Tail, Friends2, A).

% check to see if we have the same number of people in G and H
sameNumberOfNames([], []).
sameNumberOfNames([_|GTail], [_|HTail]) :-
```

```
    sameNumberOfNames(GTail, HTail).

getNames([], []).
getNames([Head|Tail], [X|L]) :-
    equal(Head, person(X,_)),
    getNames(Tail, L).
```

B Tests Listing

```
:-include(twitbook).
% Tests
:- begin_tests(twitbook).

% Imported to run tests
:- use_module(library(plunit)).

% Graphs for testing
g1([person(kara, [barry, clark]),
    person(bruce, [clark, oliver]),
    person(barry, [kara, oliver]),
    person(clark, [oliver, kara]),
    person(oliver, [kara])]).

g2([person(batman, [green_arrow, superman]),
    person(green_arrow, [supergirl]),
    person(supergirl, [flash, superman]),
    person(flash, [green_arrow, supergirl]),
    person(superman, [green_arrow, supergirl])]).

test_likes :-
    g1(G), g2(H),
    likes(G, kara, clark),
    likes(G, barry, kara),
    \+ likes(G, clark, bruce),
    likes(H, batman, green_arrow),
    \+ likes(H, supergirl, batman),
    write('Likes tests'), nl.

test_dislikes :-
    g1(G), g2(H),
```

```
    dislikes(G, kara, oliver),
    dislikes(G, oliver, barry),
    \+ dislikes(G, clark, kara),
    dislikes(H, green_arrow, superman),
    \+ dislikes(H, supergirl, superman),
    write('Dislikes tests'), nl.

test_popular :-
    g1(G), g2(H),
    popular(G, kara),
    \+ popular(G, clark),
    \+ popular(G, bruce),
    popular(H, supergirl),
    \+ popular(H, batman),
    \+ popular(H, superman),
    write('Popular tests'), nl.

test_outcast :-
    g1(G), g2(H),
    outcast(G, bruce),
    \+ outcast(G, kara),
    outcast(H, batman),
    \+ outcast(H, supergirl),
    write('Outcast tests'), nl.

test_admires :-
    g1(G), g2(H),
    admires(G, bruce, kara),
    admires(G, bruce, barry),
    admires(G, bruce, clark),
    \+ admires(G, barry, bruce),
    \+ admires(G, kara, bruce),
    admires(H, batman, supergirl),
    admires(H, batman, flash),
    admires(H, batman, superman),
    \+ admires(H, flash, batman),
    \+ admires(H, supergirl, batman),
    write('Admires tests'), nl.

test_indifferent :-
    g1(G), g2(H),
```

```
    indifferent(G, kara, bruce),
    indifferent(G, clark, bruce),
    \+ indifferent(G, bruce, kara),
    indifferent(H, green_arrow, batman),
    indifferent(H, flash, batman),
    \+ indifferent(H, batman, superman),
    write('Indifferent tests'), nl.

a1([(kara,supergirl),(bruce,batman),(barry,flash),(clark,superman),(c

test_same_world1 :-
    g1(G), g2(H),
    same_world(G,H,A),
    member((kara,supergirl), A),
    write('Same world tests 1'), nl.

test_same_world2 :-
    g1(G), g2(H),
    same_world(H,G,A),
    member((batman, bruce), A),
    write('Same world tests 2'), nl.

test_same_world3 :-
    g1(G),
    same_world(G,G,A),
    member((clark, clark), A),
    write('Same world tests 3'), nl.

test_all :-
    write('Start with tests'), nl,
    test_likes,
    test_dislikes,
    test_popular,
    test_outcast,
    test_admires,
    test_indifferent,
    test_same_world1,
    test_same_world2,
    test_same_world3,
    write('All tests have been run'), nl.
```

```
:- end_tests(twitbook).
```