Haskell intro

Assignment 2

Kai Arne S. Myklebust, Silvan Adrian

Handed in: October 2, 2018

Contents

1 Design/Implementation

We implemented all needed predicates by not using the Prolog out of predicates, which we were restricted to use. So we had to implement predicates for select or not ourselves.

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.

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)]).

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

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

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

/* 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, _).

/* */
selectList(X, [X|Tail], Tail).
selectList(Elem, [Head|Tail], [Head|Rest]) :-
    selectList(Elem, Tail, Rest).

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).

getFriends([person(X, Friends)|_], X, Friends).
getFriends([_|T], X, Friends) :- getFriends(T, X, Friends).

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

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

% Task d
outcast(G, X) :-
    isMember(person(X, Friends), G),
    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) :-
    % notLikes(G, X, Y),
    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) :-
```

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_all :-
    write('Start with tests'), nl,
    test_likes,
    test_dislikes,
    test_popular,
    test_outcast,
    test_admires,
    write('All tests have been run'), nl.

:- end_tests(twitbook).
```