HY487
Assignment 4

Γιώργος Γεραμούτσος
csd3927

Task A

The rule interconnected(X, Y):- interconnected(Y, X) would be problematic for representing a non-directed graph. Since this rule is recursive it means that it refers to itself in its definition. This means that when a programme would try to satisfy the rule it would try to find a solution for interconnected(Y, X) before it can determine the solution for interconnected(X, Y) and this would lead to an infinite loop. Using the 2 rules would be a better way to represent the bi-directional relationship because they can be satisfied individually. In some situations the recursive single rule could be used if in a controlled manner. For example if used in conjunction with another rule that would cause it to break out of the recursive calls at some point.

Note that Task B and C are implemented for the non-directed graph in figure 1b

Task B

SWISH File Edit Examples Help

Program

```prolog
interconnected(X, Y) :- connect(X, Y).
interconnected(X, Y) :- connect(Y, X).

connect(a, b).
connect(b, c).
connect(b, d).
connect(c, f).
connect(d, c).
connect(f, e).
connect(e, d).

exists_path(Inital, Goal) :-
    Inital = Goal,
    !.
exists_path(Inital, Goal) :-
    interconnected(Inital, Goal),
    !.
exists_path(Inital, Goal) :-
    (interconnected(Inital, Next) ; interconnected(Goal, Prev)),
    exists_path(Next, Prev).
```

exists_path(a,f).

true
true
true

exists_path(f,a).

true
true
true

exists_path(e,c).

true
true
true
true
true

exists_path(b,e).

true
true
true
true
true

Task C

```
%define the facts and rules
interconnected(X, Y) :- connect(X, Y).
interconnected(X, Y) :- connect(Y, X).

connect(a, b).
connect(b, c).
connect(b, d).
connect(c, f).
connect(d, c).
connect(f, e).
connect(e, d).


path(Initial, Goal, Route) :-
    path_aux(Initial, Goal, [Goal], Route).

path_aux(Initial, Goal, Visited, [Initial|Visited]) :-
    interconnected(Initial, Goal).

path_aux(Initial, Goal, Visited, Route) :-
    interconnected(Intermediate, Goal),
    not(member(Intermediate, Visited)),
    Intermediate \== Initial,

    path_aux(Initial, Intermediate, [Intermediate| Visited], Route).

```

path(f,a,Route)

Route = [f, c, b, a]
Route = [f, e, d, c, b, a]
Route = [f, e, d, b, a]
Route = [f, c, d, b, a]

path(a,f,Route)

Route = [a, b, c, f]
Route = [a, b, d, c, f]
Route = [a, b, d, e, f]
Route = [a, b, c, d, e, f]

path(b,e,Route)

Route = [b, c, f, e]
Route = [b, d, c, f, e]
Route = [b, d, e]
Route = [b, c, d, e]

path(c,e,Route)

Route = [c, f, e]
Route = [c, d, e]
Route = [c, b, d, e]

Task D

```
%define the facts and rules
interconnected(X, Y, C) :- connect(X, Y, C).
interconnected(X, Y, C) :- connect(Y, X, C).

connect(a, b, 3).
connect(b, c, 12).
connect(b, d, 4).
connect(c, f, 11).
connect(d, c, 7).
connect(f, e, 15).
connect(e, d, 5).

% cost_path rule that uses the auxiliary rule to account for visited nodes and path cost
cost_path(Initial, Goal, Route, Cost) :-
    cost_path_aux(Initial, Goal, [Goal], 0, Route, Cost).

% auxiliary cost_path rule accounting for visited nodes and path cost
cost_path_aux(Initial, Goal, Visited, CostSoFar, [Initial|Visited], CurrentCost) :-
    interconnected(Initial, Goal, Cost),
    CurrentCost is CostSoFar + Cost.

cost_path_aux(Initial, Goal, Visited, CurrentCost, Route, TotalCost) :-
    interconnected(Intermediate, Goal, Cost), % Check if intermediate node exists and get its cost
    not(member(Intermediate, Visited)), % Check if intermediate not already visited
    Intermediate \== Initial,
    NewCost is CurrentCost + Cost, % Update the current path cost
    % Add Intermediate node in visited list and recursively call
    cost_path_aux(Initial, Intermediate, [Intermediate| Visited], NewCost, Route, TotalCost).
```

cost_path(a,f,Route,Cost).

Cost = 26,
Route = [a, b, c, f]
Cost = 25,
Route = [a, b, d, c, f]
Cost = 27,
Route = [a, b, d, e, f]
Cost = 42,
Route = [a, b, c, d, e, f]

cost_path(b,e,Route,Cost).

Cost = 38,
Route = [b, c, f, e]
Cost = 37,
Route = [b, d, c, f, e]
Cost = 9,
Route = [b, d, e]
Cost = 24,
Route = [b, c, d, e]

cost_path(f,a,Route,Cost).

Cost = 26,
Route = [f, c, b, a]
Cost = 42,
Route = [f, e, d, c, b, a]
Cost = 27,
Route = [f, e, d, b, a]
Cost = 25,
Route = [f, c, d, b, a]

cost_path(a,e,Route,Cost).

Cost = 41,
Route = [a, b, c, f, e]
Cost = 40,
Route = [a, b, d, c, f, e]
Cost = 12,
Route = [a, b, d, e]
Cost = 27,
Route = [a, b, c, d, e]