CS-6374: Computational Logic

Final Exam

Abhijit Balaji - AXB190013

Problem 1: Graph coloring using CLPFD

Solution:

- I used the Test and generate paradigm in CLPFD to solve this problem
- First I wrote a predicate generate_node_colors which returns a list of all possible combinations of nodes and colors of format [(Node1, Color1), (Node2, Color2), ...]
- Then I bind the colors to domain 10,11,12,13 (10 for ‘r’, 11 for ‘g’) etc.
- Then I wrote another predicate **color_different** which is False if adjacent nodes have the same color.
- Then I wrote another predicate **map_index2color** to map 10 to “r”, 11 to “g”, 12 to “b”, 13 to “y”.

```
:- use_module(library(clpfd)).

% modified append to return a list when empty list is given
append([],L,[L]).
append([X|T], Y, [X|Z]) :- append(T,Y,Z).

graphcolor(Nodes, Edges, NodeColorsOutput) :-
   generate_node_colors(Nodes, Colors, NodeColors),
   Colors ins 10..13, % create all domains
   color_different(Edges, NodeColors),
   labeling([ff], Colors),
   map_index2color(NodeColors, [], NodeColorsOutput).


% Generate all possible node colorings
generate_node_colors([],[],[]).
generate_node_colors([N|Nodes], [C|Colors], [(N, C)|NodeColors]) :-
   generate_node_colors(Nodes, Colors, NodeColors).

% Test if adjacent colors are different
color_different([], _).
color_different([(N1, N2)|Edges], Colors) :-
```

```
    member((N1, C1), Colors),
    member((N2, C2), Colors),
    C1 #\= C2,
    color_different(Edges, Colors).

% maps the integer colors to letters
map_index2color([], Acc, Output) :- Output = Acc.
map_index2color([(N, C)|NodeColors], Acc, Output):-
    (
        C == 10 -> append(Acc, (N, 'r'), Output1);
        C == 11 -> append(Acc, (N, 'g'), Output1);
        C == 12 -> append(Acc, (N, 'b'), Output1);
        append(Acc, (N, 'y'), Output1)
    ),
    map_index2color(NodeColors, Output1, Output).
```

Output:

```
abhijit@dwave ~/studies/logic_programming/exams/finals (main)$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult("q1.pl").
true.

?- graphcolor([1,2,3,4,5],[(1,2),(1,3),(2,4),(2,5),(3,4),(3,5)], A).
A = [(1, r),  (2, g),  (3, g),  (4, r),  (5, r)] ;
A = [(1, r),  (2, g),  (3, g),  (4, r),  (5, b)] ;
A = [(1, r),  (2, g),  (3, g),  (4, r),  (5, y)] .

?- graphcolor([1, 2, 3], [(1,2),(1,3)], N).
N = [(1, r),  (2, g),  (3, g)] ;
N = [(1, r),  (2, g),  (3, b)] ;
N = [(1, r),  (2, g),  (3, y)] .

?- graphcolor([1,2,3,4,5,6,7],[(1,2),(1,3),(2,4),(2,5),(3,6),(3,7)], A).
A = [(1, r),  (2, g),  (3, g),  (4, r),  (5, r),  (6, r),  (7, r)] ;
A = [(1, r),  (2, g),  (3, g),  (4, r),  (5, r),  (6, r),  (7, b)] ;
A = [(1, r),  (2, g),  (3, g),  (4, r),  (5, r),  (6, r),  (7, y)] .

?-
```

Problem 2:

```
sub([],X).
sub([X|L], L1) :- sff([X|L2], L1), sub(L, L2).

sff(L, L).
sff(L1, [X|L]) :- sff(L1, L).

comm(X,Y) :- sub([X,Y] ,[a,t,s]), sub([X,Y] ,[a,s,t]).
```

A. What does the predicate sff do?

Ans: The predicate sff(A, B) returns the common tail between A and B where len(B) >= len(A). Thus

- sff([11,12,13,14,15],[1,2,3,4,5,11,12,13,14,15]) is **True.**
- *sff([11,12,13,14,15],[1,2,3,4,5,11,12,13,14]) is* **False**

B. Show the search tree for query ?- sub([X,Y], [a,t,s]).

Ans:

```
[trace]  ?- sub([X,Y], [a,t,s]).
   Call: (10) sub([_15704, _15710], [a, t, s]) ? creep
   Call: (11) sff([_15704|_16182], [a, t, s]) ? creep
   Exit: (11) sff([a, t, s], [a, t, s]) ? creep
   Call: (11) sub([_15710], [t, s]) ? creep
   Call: (12) sff([_15710|_16320], [t, s]) ? creep
   Exit: (12) sff([t, s], [t, s]) ? creep
   Call: (12) sub([], [s]) ? creep
   Exit: (12) sub([], [s]) ? creep
   Exit: (11) sub([t], [t, s]) ? creep
   Exit: (10) sub([a, t], [a, t, s]) ? creep
X = a,
Y = t
```

Basically, The solutions are:

- X=a, Y=t;
- X=a, Y=s;
- X=t, Y=s;

Basically it returns all combinations of tuples of length 2.

I.e len(given _list)C2 (C is the combinations formula)

C. Show the search tree for query ?- sub([X,Y], [a,t,s]).

Similar to above:

```
[trace]  ?- sub([X,Y], [a,s,t]).
   Call: (10) sub([_11940, _11946], [a, s, t]) ? creep
   Call: (11) sff([_11940|_12418], [a, s, t]) ? creep
   Exit: (11) sff([a, s, t], [a, s, t]) ? creep
   Call: (11) sub([_11946], [s, t]) ? creep
   Call: (12) sff([_11946|_12556], [s, t]) ? creep
   Exit: (12) sff([s, t], [s, t]) ? creep
   Call: (12) sub([], [t]) ? creep
   Exit: (12) sub([], [t]) ? creep
   Exit: (11) sub([s], [s, t]) ? creep
   Exit: (10) sub([a, s], [a, s, t]) ? creep
X = a,
Y = s
```

Similar to above the solutions are:

- X=a, Y=s;
- X=a, Y=t;
- X=s, Y=t.

D. Output of ?- comm(A,B).

Ans: This finds the common tuples of length 2 which is generated by the predicate sub using [a,t,s] and [a,s,t].
Thus outputs are:

- A = a, B = t;
- A = a, B = s.

Problem 3: Define the following terms

1. Definite Clause Grammar:

Ans:
- It is basically a notation to represent Context-Free Grammar in Prolog.
- Example:
    - sentence --> noun_phrase, verb_phrase.
    - noun_phrase --> determiner, noun.
- Basically DCG is just a syntactic sugar for normal definite clauses.

For example:
The DCG:

sentence --> noun_phrase, verb_phrase.

Is the syntactic sugar representation of:

sentence(S1,Output) :- noun_phrase(S1,X), verb_phrase(X,Output)

2. Coinductive Logic Programming

Ans:
- Coinduction is basically the dual of induction.
- Inductive definition has 3 components: initiality, iteration and minimality.
- Coinduction eliminates the initiality condition (base case) and replaces the minimality condition with maximality.
- Basically Co-inductive logic programming is an extension of logic programming where the proofs may be of infinite length.
- Basically it allows predicates such as:

    generate_integer([H|T]) :- integer(H), generate_integer(T).

Without defining the base case thereby resulting in the proof being of infinite length and also the answer being of infinite length.

3. Gelfond-Lifschitz:

Ans: It is the method for finding Answer sets / worlds especially in cyclical programs
1. Given an answer set S (We guess it), for each p in S, delete all rules whose body contains “not p”
2. Delete all goals of the form “not q” in remaining goals
3. Compute the least fixed point, L, of the residual program
4. If S=L then S is an answer set.

Problem 4: No Edges In

- For every node in the given nodes, check if it is **not** present as the second element of the tuple in the edges list

```
no_edge_in(Nodes,Edges,N):-
    select(N,Nodes,_),
    -member([_,N],Edges).

member(X,[X|_]).
member(X,[_|Tail]):- member(X,Tail).
% CWA
-member(X,L) :- not member(X,L).

select(X,[X|Xs],Xs).
select(X,[Y|Ys],[Y|Zs]):- select(X,Ys,Zs).
```

```
abhijit@dwave ~/studies/logic_programming/exams/finals (main)$ scasp -i q4.lp

?- no_edge_in([1, 2, 3], [[1,2],[1,3]], N).
% QUERY:?- no_edge_in([1,2,3],[[1,2],[1,3]],Var0).

        ANSWER: 1 (in 4.872 ms)

MODEL:
{ no_edge_in([1,2,3],[[1,2],[1,3]],1),  select(1,[1,2,3],[2,3]),  -member([Va
r1,1],[[1,2],[1,3]]),  not member([Var1,1],[[1,2],[1,3]]),  not member([Var1,
1],[[1,3]]),  not member([Var1,1],[]),  not -member([Var1,1],[[Var1,1]|Var2])
,  member([Var1,1],[[Var1,1]|Var2]) }

BINDINGS:
Var0 = 1 ?
```

```
abhijit@dwave ~/studies/logic_programming/exams/finals (main)$ scasp -i q4.lp

?- no_edge_in([1,2,3,4,5,6],[[1,2],[1,3],[2,4],[2,5],[3,4],[3,5]], N).
% QUERY:?- no_edge_in([1,2,3,4,5,6],[[1,2],[1,3],[2,4],[2,5],[3,4],[3,5]],Var
0).

        ANSWER: 1 (in 14.676 ms)

MODEL:
{ no_edge_in([1,2,3,4,5,6],[[1,2],[1,3],[2,4],[2,5],[3,4],[3,5]],1),  select(
1,[1,2,3,4,5,6],[2,3,4,5,6]),  -member([Var1,1],[[1,2],[1,3],[2,4],[2,5],[3,4
],[3,5]]),  not member([Var1,1],[[1,2],[1,3],[2,4],[2,5],[3,4],[3,5]]),  not
member([Var1,1],[[1,3],[2,4],[2,5],[3,4],[3,5]]),  not member([Var1,1],[[2,4]
,[2,5],[3,4],[3,5]]),  not member([Var1,1],[[2,5],[3,4],[3,5]]),  not member(
[Var1,1],[[3,4],[3,5]]),  not member([Var1,1],[[3,5]]),  not member([Var1,1],
[]),  not -member([Var1,1],[[Var1,1]|Var2]),  member([Var1,1],[[Var1,1]|Var2]
) }

BINDINGS:
Var0 = 1 ?
```

Problem 4: Alternate solution - This is much easier to write in prolog

```
no_edge_in(Nodes,Edges,N):-
    select(N,Nodes,_),
    \+ member([_,N],Edges).
```

```
abhijit@dwave ~/studies/logic_programming/exams/finals (main)$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult("q4.pl").
true.

?- no_edge_in([1, 2, 3], [(1,2),(1,3)], N).
N = 1 ;
false.

?- no_edge_in([1,2,3,4,5,6],[(1,2),(1,3),(2,4),(2,5),(3,4),(3,5)], N).
N = 1 ;
N = 6.

?- 
```

Problem 5

A. Define Abduction

Ans: In a nutshell abduction is assumption based reasoning. From the example given in the class, deduction is basically for every X if it is a swan(X) => it is white(X). Abduction is the reverse of this. I.e when we have white(10), then we can abduce that it is a swan(10). Thus abduction allows us to do assumption based reasoning. Basically, it means forming logical conclusions using what is known.

swan(X) :- not -swan(X).
-swan(X) :- not swan(X).
white(X) :- swan(X)

The above rules say that either the given swan(X) is either True/False.
Thus, when we query: white(10), In a world where swan(10) is True we can abduce that white(10) is True.

B. Dual Rules

```
% Question
p :- q, not r.
p :- q, not s, not t.
s :- not p.
r :- not p.
v :- not u.
% Compute the dual rules for p, s and r.
%------------------------------------------------

%% Answer
% Dual for p
p :- (q, not r) ; (q, not s, not t).
% negating p using Demorgan's law
not_p :- (not q ; r), (not q ; s ; t).

% Splitting into multiple rules
not_p :- not q. % rule 1
not_p :- not q, s. % rule 1 makes this rule redundant.
not_p :- not q, t. % rule 1 makes this rule redundant.
not_p :- r, not q.
not_p :- r, s.
not_p :- r, t.

% Dual for s
```

```
not_s :- p.

% Dual for r
not_r :- p.
```

C. Abductibles in sCasp

exams > finals > q5.lp

```
p :- q, not r.
p :- q, not s, not t.
s :- not p.
r :- not p.
v :- not u.

% adding even loops
% even loop for q
q :- not not_q.
not_q :- not q.
% even loop for t
t :- not not_t.
not_t :- not t.
% even loop for u
u :- not not_u.
not_u :- not u.
% even loop for v
v :- not not_v.
not_v :- not v.
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
abhijit@dwave ~/studies/logic_programming/exams/finals (main)$ scasp -i q5.lp

?- not s.
% QUERY:?- not s.

        ANSWER: 1 (in 1.034 ms)

MODEL:
{ not s,  p,  q,  not not_q,  not r }

BINDINGS: ?
```

Problem 6: Lights out

Algorithm: Chasing the lights - For 5X5

- Look at the first row and see which positions are turned on
- Toggle the switches in the 2nd row where the 1st row light is on
- Repeat this until the last row - This is called as chasing the lights.
- This will result in a grid where only the lights are turned on only in the last row
- For a 5X5 board we will arrive at these 7 cases.

Columns on in the bottom row	Column to push on the top row
1,2,3	2
1,2,4,5	3
1,3,5	5
1,5	1,2
2,3,5	1
2,4	1,4
3,4,5	4

- Then repeat the same chasing the light process.
- But this time when you do it at the end you will have all the lights off.

```
% modified append to return a list when empty list is given
append([],L,[L]).
append([X|T], Y, [X|Z]) :- append(T,Y,Z).
% This solves only the 5X5 puzzle
% everything using 1 indexing
% removes the element
remove_at(X,[X|Xs],1,Xs).
remove_at(X,[Y|Xs],K,[Y|Ys]) :- K > 1,
  K1 is K - 1, remove_at(X,Xs,K1,Ys).
% insert an element X at position K in the list L and the result is R
insert_at(X,L,K,R) :- remove_at(X,R,K,L).
% Modify the element at position K with X in the list  L and the
result is R
modify_at(X,L,K,R) :- remove_at(_, L, K, R1), insert_at(X, R1, K, R).

% index into 2D list
% Uses 1 indexing.
get_index2D(Grid, (RowIndex, ColIndex), Element):-
   nth1(RowIndex, Grid, Row),
   nth1(ColIndex, Row, Element).

% modifies elements in a given 2D index.
% This Modifies the element at position (R,C) with X in the 2D Grid
and the result is OutGrid
modify_index2D(X, Grid, (RowIndex, ColIndex), OutGrid):-
   remove_at(Row, Grid, RowIndex, RemovedGrid), % get the row
   modify_at(X, Row, ColIndex, ModifiedRow), % modify the row
   insert_at(ModifiedRow, RemovedGrid, RowIndex, OutGrid).

flip_element(Grid, (R,C), FlippedGrid):-
   % does not do any check for index. So check if the index is valid
before calling this function
   get_index2D(Grid, (R,C), Element),
   (Element == 0 -> modify_index2D(1, Grid, (R,C), FlippedGrid);
modify_index2D(0, Grid, (R,C), FlippedGrid)).

check_index(Grid, (R, C)) :-
   length(Grid, MyLen),
   R >= 1, R =< MyLen,
   C >= 1, C =< MyLen.
```

```
flip_one_switch(Grid, (R,C), FlippedGrid):-
   check_index(Grid, (R,C)) -> flip_element(Grid, (R,C),
FlippedGrid); FlippedGrid = Grid.

% This flips the Grid as given in problem definition
% The given R,C and all the 4 adjacent cells are flipped
flip_switch(Grid, (R,C), FlippedGrid) :-
   % writeGrid(Grid),
   % flip that position
   flip_one_switch(Grid, (R,C), FG1),
   R1 is R - 1, R2 is R + 1,
   C1 is C - 1, C2 is C + 1,
   % left flip
   flip_one_switch(FG1, (R,C1), FG2),
   % right flip
   flip_one_switch(FG2, (R,C2), FG3),
   % top flip
   flip_one_switch(FG3, (R1,C), FG4),
   % bottom flip
   flip_one_switch(FG4, (R2,C), FlippedGrid).
   % nl,
   % writeGrid(FlippedGrid).

% driver predicate
get_on_switch_in_row(RowList, OutList):-
get_on_switch_in_row(RowList, 1, [], OutList).
get_on_switch_in_row(RowList, C, Acc, OutList):-
   length(RowList, MyLen),
   C =< MyLen,
   nth1(C, RowList, Element),
   (Element == 1 -> append(Acc, C, Acc1); Acc1 = Acc),
   C1 is C+1,
   get_on_switch_in_row(RowList, C1, Acc1, OutList).
% base case
get_on_switch_in_row(RowList, C, Acc, OutList):-
   length(RowList, MyLen),
   C > MyLen,
   OutList = Acc.

% base case
% flips the  switches in the given row of the grid using a list of
```

```
columns
flip_row(Grid, _, [], FlippedGrid) :- FlippedGrid = Grid.
flip_row(Grid, R, [C|ColList], FlippedGrid):-
   writeMove((R,C)),
   flip_switch(Grid, (R,C), FlippedGrid1),
   flip_row(FlippedGrid1, R, ColList, FlippedGrid).

% driver code
solve_until_last_row(Grid, FlippedGrid):-
   solve_until_last_row(Grid, 1, FlippedGrid).
solve_until_last_row(Grid, 5, FlippedGrid) :- FlippedGrid = Grid. %
stop when you reach the last row
solve_until_last_row(Grid, RowIndex, FlippedGrid):-
   RowIndex<5,
   % get all columns with switch on in that row
   nth1(RowIndex, Grid, Row),
   get_on_switch_in_row(Row, ColList),
   RowIndex1 is RowIndex + 1,
   flip_row(Grid, RowIndex1, ColList, FG),
   solve_until_last_row(FG, RowIndex1, FlippedGrid).


solve_last_row(Grid, FinalAns) :-
   % The seven cases are listed above putting an if else for the
cases
   nth1(5, Grid, LastRow),
   get_on_switch_in_row(LastRow, ColList),
   % Flip the corresponding first  row
   (
       ColList == [1,2,3] -> flip_switch(Grid, (1,2), FlippedGrid),
writeMove((1,2));
       ColList == [1,2,4,5] -> flip_switch(Grid, (1,3), FlippedGrid),
writeMove((1,3));
       ColList == [1,3,4] -> flip_switch(Grid, (1,5), FlippedGrid),
writeMove((1,5));
       ColList == [1,5] -> flip_switch(Grid, (1,1), FlippedGrid1),
writeMove((1,1)), flip_switch(FlippedGrid1, (1,2), FlippedGrid),
writeMove((1,2));
       ColList == [2,3,5] -> flip_switch(Grid, (1,1), FlippedGrid),
writeMove((1,1));
       ColList == [2,4] -> flip_switch(Grid, (1,1), FlippedGrid1),
writeMove((1,1)), flip_switch(FlippedGrid1, (1,4), FlippedGrid),
```

```
writeMove((1,4));
       ColList == [3,4,5] -> flip_switch(Grid, (1,4), FlippedGrid),
writeMove((1,4));
       FlippedGrid = Grid
   ),
   % repeat solve until last Row
   solve_until_last_row(FlippedGrid, FinalAns).

% The predicate to use
lights_out(Grid):-
   write("Given Grid: "), nl,
   writeGrid(Grid),
   write("The (Rows, Cols) to flip are given below: "), nl,
   solve_until_last_row(Grid, FlippedGrid),
   solve_last_row(FlippedGrid, FinalAns),
   write("The grid after following the above procedure: "), nl,
   writeGrid(FinalAns).




% write the moves
writeMove((R,C)):- write(R), write(", "), write(C), nl.
% writes a given 2D grid to console
writeGrid([R1, R2, R3, R4, R5|_]):-
   writeRow(R1),
   writeRow(R2),
   writeRow(R3),
   writeRow(R4),
   writeRow(R5).

% writes a given row to the console. This is used by write grid
writeRow([A,B,C,D,E|_]):-
   write(A), write(" "), write(B), write(" "), write(C), write(" "),
write(D), write(" "), write(E),nl.
```

Output:

```
abhijit@dwave ~/studies/logic_programming/exams/finals (main)$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult("q6.pl").
true.

?- lights_out([[0,0,0,0,1],[1,1,1,1,1],[0,0,0,1,0],[0,1,1,1,0],[0,1,0,0,1]]).
Given Grid:
0 0 0 0 1
1 1 1 1 1
0 0 0 1 0
0 1 1 1 0
0 1 0 0 1
The (Rows, Cols) to flip are given below:
2, 5
3, 1
3, 2
3, 3
4, 2
4, 5
5, 2
5, 3
5, 5
1, 1
1, 2
2, 3
3, 1
3, 3
3, 4
4, 1
4, 3
4, 5
5, 4
5, 5
The grid after following the above procedure:
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
true .

?- 
```

```
abhijit@dwave ~/studies/logic_programming/exams/finals (main)$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult("q6.pl")
|    .
true.

?- lights_out([[0,1,1,1,1],[0,1,1,0,1],[1,0,1,1,0],[1,1,0,1,1],[0,1,1,1,1]]).
Given Grid:
0 1 1 1 1
0 1 1 0 1
1 0 1 1 0
1 1 0 1 1
0 1 1 1 1
The (Rows, Cols) to flip are given below:
2, 2
2, 3
2, 4
2, 5
3, 1
3, 2
3, 4
3, 5
4, 1
4, 2
4, 5
5, 3
5, 4
5, 5
1, 2
2, 1
2, 2
2, 3
3, 4
4, 1
4, 2
4, 4
4, 5
5, 4
The grid after following the above procedure:
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
true .

?- 
```

```
abhijit@dwave ~/studies/logic_programming/exams/finals (main)$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult("q6.pl").
true.

?- lights_out([[1,0,0,0,1],[1,0,0,1,1],[1,0,0,1,0],[0,1,1,1,0],[0,1,0,0,1]]).
Given Grid:
1 0 0 0 1
1 0 0 1 1
1 0 0 1 0
0 1 1 1 0
0 1 0 0 1
The (Rows, Cols) to flip are given below:
2, 1
2, 5
3, 2
4, 1
4, 2
4, 3
4, 4
4, 5
5, 2
1, 1
1, 4
2, 1
2, 2
2, 3
2, 4
2, 5
3, 1
3, 2
3, 3
4, 1
4, 3
4, 5
5, 2
5, 5
The grid after following the above procedure:
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
true .

?-
```