

University of Texas at Dallas

CS 6374 - Computational Logic

Homework 05

**Problem 1:** Program the N-Queen problem in CLP(FD). Follow the structure of the 8 queens problem whose code is given to you.

Ans.

```
n_queens(N, Qs) :-
    length(Qs, N),
    Qs ins 1..N,
    safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :-
    safe_queens(Qs, Q, 1),
    safe_queens(Qs).

safe_queens([], _, _).
safe_queens([Q|Qs], Q0, D0) :-
    Q0 #\= Q,
    abs(Q0 - Q) #\= D0,
    D1 #= D0 + 1,
    safe_queens(Qs, Q0, D1).

% n_queens(4,Q), label(Q).
```

```

22 X #= Y+N; X #= Y-N.
23 attack(X,N,[_|Ys]) :-
24     N1 #= N+1, attack(X,N1,Ys).
25
26 % permutation(L1,P): P is a permutation of L1
27 permutation([],[]).
28 permutation(Xs,[Z|Zs]) :-
29     select(Z,Xs,Ys), permutation(Ys,Zs).
30
31 % range(M,N,List): List is list of integers [M,N], M<N.
32 range(N,N,[N]).
33 range(M,N,[M|Ns]) :-
34     M #< N, M1 #= M+1, range(M1,N,Ns).
35
36 % select(X,List,R): R #= a List with one removed X
37 select(X,[X|T],T).
38 select(X,[H|T],[H|R]) :- %X \= H,
39     select(X,T,R).
40
41 %-----
42 n_queens(N, Qs) :-
43     length(Qs, N),
44     Qs ins 1..N,
45     safe_queens(Qs).
46
47 safe_queens([]).
48 safe_queens([Q|Qs]) :-
49     safe_queens(Qs, Q, 1), safe_queens(Qs).
50
51 safe_queens([], _, _).
52 safe_queens([Q|Qs], Q0, D0) :-
53     Q0 #\= Q,
54     abs(Q0 - Q) #\= D0,
55     D1 #= D0 + 1,
56     safe_queens(Qs, Q0, D1).
57
58 % n_queens(4,Q), label(Q).
59
?- ['hw5q1.lp'].
true.

?- queens(4,Q).
Q = [2, 4, 1, 3] ;
Q = [3, 1, 4, 2] ;
false.

?- n_queens(4,Q), label(Q).
Q = [2, 4, 1, 3] ;
Q = [3, 1, 4, 2] ;
false.

?- n_queens(5,Q), label(Q).
Q = [1, 3, 5, 2, 4] ;
Q = [1, 4, 2, 5, 3] ;
Q = [2, 4, 1, 3, 5] ;
Q = [2, 5, 3, 1, 4] ;
Q = [3, 1, 4, 2, 5] ;
Q = [3, 5, 2, 4, 1] ;
Q = [4, 1, 3, 5, 2] ;
Q = [4, 2, 5, 3, 1] ;
Q = [5, 2, 4, 1, 3] ;
Q = [5, 3, 1, 4, 2] ;
false.

?- n_queens(3,Q), label(Q).
false.

?-

```

**Problem 2:** Write a CLP(FD) program to solve cryptarithmic addition problems.

**Ans.**

```
genCrypt([H1|T1] + [H2|T2] = [H3|T3]) :-
    append([H1|T1], [H2|T2], Temp),
    append(Temp, [H3|T3], Duplicates),

    list_to_set(Duplicates, DistictSet),
    DistictSet ins 0..9,
    all_distinct(DistictSet),

    H1 #> 0, H2 #> 0, H3 #> 0,

    value([H1|T1], S1),
    value([H2|T2], S2),
    value([H3|T3], S3),
    S3 #= S1 + S2,

    labeling([], DistictSet).

value([], 0).
value([H|T], V) :-
    length(T, L), Base #= 10^L,
    BaseValue #= Base * H,

    V #= V1 + BaseValue,
    value(T, V1).
```

The screenshot shows a Prolog IDE with three panes. The left pane displays the source code for the `genCrypt` program. The middle pane shows the execution results for a query: `?- genCrypt([A,M] + [P,M] = [D,A,Y]).` The results show a list of solutions for A, M, P, D, and Y. The right pane shows the execution results for a query: `?- genCrypt([E,A,T] + [T,H,A,T] = [A,P,P,L,E]).` The results show a list of solutions for E, A, T, H, P, L, and S.

```
1 % CS6374: Computational Logic - Homework 05
2 % Rahul Nalawade [RSN170330]
3 % Date: 2018-03-26
4
5 :- use_module(library(clpfd)).
6
7 %----- Q2 -----
8 genCrypt([H1|T1] + [H2|T2] = [H3|T3]) :-
9     append([H1|T1], [H2|T2], Temp),
10    append(Temp, [H3|T3], Duplicates),
11
12    list_to_set(Duplicates, DistictSet),
13    DistictSet ins 0..9,
14    all_distinct(DistictSet),
15
16    H1 #> 0, H2 #> 0, H3 #> 0,
17
18    value([H1|T1], S1),
19    value([H2|T2], S2),
20    value([H3|T3], S3),
21    S3 #= S1 + S2,
22
23    labeling([], DistictSet).
24
25 value([], 0).
26 value([H|T], V) :-
27     length(T, L), Base #= 10^L,
28     BaseValue #= Base * H,
29
30     V #= V1 + BaseValue,
31     value(T, V1).
32
33 %-----
34
```

```
?- genCrypt([A,M] + [P,M] = [D,A,Y]).
true.
A = 2,
M = 5,
P = 9,
D = 1,
Y = 0 ;
A = 2,
M = 7,
P = 9,
D = 1,
Y = 4 ;
A = 2,
M = 8,
P = 9,
D = 1,
Y = 6 .

?- genCrypt([S,E,N,D] + [M,O,R,E] = [M,O,N,E,Y]).
S = 9,
E = 5,
N = 6,
D = 7,
M = 1,
O = 0,
R = 8,
Y = 2 ;
false.

?- genCrypt([L,I,V,E] + [V,I,L,E] = [E,V,I,L]).
L = 8,
I = 0,
V = 1,
E = 9.

?- []
true.
```

```
?- genCrypt([E,A,T] + [T,H,A,T] = [A,P,P,L,E]).
true.
E = 8,
A = 1,
T = 9,
H = 2,
P = 0,
L = 3.

?- genCrypt([B,I,K,E] + [R,I,D,E] = [R,O,A,D,S]).
B = 8,
I = 6,
K = 9,
E = 7,
R = 1,
D = 2,
O = 0,
A = 3,
S = 4 ;
B = 8,
I = 6,
K = 9,
E = 7,
R = 1,
D = 5,
O = 0,
A = 3,
S = 4 .

?- genCrypt([D,O,N,A,L,D] + [G,E,R,A,L,D] = [R,O,B,E,R,T]).
D = 5,
O = 2,
N = 6,
A = 4,
L = 8,
G = 1,
E = 9,
R = 7,
B = 3,
T = 0 ;
false.

?- []
```

**Problem 3:** Program the Zebra puzzle in CLP(FD).

**Ans.**

```
solve(N,C,P,A,D) :-
    N ins 1..5, C ins 1..5, P ins 1..5, A ins 1..5, D ins 1..5,
    all_different(N), all_different(C), all_different(P), all_different(A),
    all_different(D),

    N = [N1,N2,N3,N4,N5], C = [C1,C2,C3,C4,C5], P = [P1,P2,P3,P4,P5],
    A = [A1,A2,A3,A4,A5], D = [D1,D2,D3,D4,D5],

    %Rules: 1. 2. 3. 4. 5.
    N1 #= C2, N2 #= A1, N3 #= P1, N4 #= D3, N5 #= 1,

    %Rules: 6. 7. 8. 9. 10.
    C1 #= D4, C1 #= C5 + 1, P5 #= A4, P2 #= C3, D5 #= 3,

    %Rules: 11. 12. 13. 14.
    N5 #= C4 + 1 #\ N5 #= C4 - 1, P3 #= D1, A3 #= P4 + 1 #\ A3 #= P4 - 1,
    A5 #= P2 + 1 #\ A5 #= P2 - 1,

    labeling([ff],N), labeling([ff],C), labeling([ff],P), labeling([ff],A),
    labeling([ff],D),

    H = [lives(englishman,N1), lives(spaniard,N2), lives(japanese,N3),
    lives(italian,N4), lives(norwegian,N5)],

    member(lives(Owner1,A2), H), write('Zebra owner: '), write(Owner1), nl,
    member(lives(Owner2,D2), H), write('Water drinker: '), write(Owner2), nl.
```

The screenshot shows a Prolog IDE with two panes. The left pane contains the Prolog code for solving the Zebra puzzle. The right pane shows the execution output, which includes the solution to the puzzle.

**Left Pane (Code):**

```
4 :- use_module(library(clpfd)).
5 %----- Q3 -----
6 solve(N,C,P,A,D) :-
7     N ins 1..5, C ins 1..5, P ins 1..5, A ins 1..5, D ins 1..5,
8     all_different(N), all_different(C), all_different(P), all_different(A),
9     all_different(D),
10
11     N = [N1,N2,N3,N4,N5], C = [C1,C2,C3,C4,C5],
12     P = [P1,P2,P3,P4,P5], A = [A1,A2,A3,A4,A5],
13     D = [D1,D2,D3,D4,D5],
14
15     %Rules: 1. 2. 3. 4. 5.
16     N1 #= C2, N2 #= A1, N3 #= P1, N4 #= D3, N5 #= 1,
17
18     %Rules: 6. 7. 8. 9. 10.
19     C1 #= D4, C1 #= C5 + 1, P5 #= A4, P2 #= C3, D5 #= 3,
20
21     %Rules: 11. 12. 13. 14.
22     N5 #= C4 + 1 #\ N5 #= C4 - 1, P3 #= D1, A3 #= P4 + 1 #\ A3 #= P4 - 1,
23     A5 #= P2 + 1 #\ A5 #= P2 - 1,
24
25     labeling([ff],N), labeling([ff],C),
26     labeling([ff],P), labeling([ff],A),
27     labeling([ff],D),
28
29     H = [lives(englishman,N1), lives(spaniard,N2), lives(japanese,N3),
30     lives(italian,N4), lives(norwegian,N5)],
31
32     member(lives(Owner1,A2), H), write('Zebra owner: '), write(Owner1), nl,
33     member(lives(Owner2,D2), H), write('Water drinker: '), write(Owner2), nl.
```

**Right Pane (Output):**

```
?- ['hw5q3.lp'].
true.

?- solve([Englishman, Spaniard, Japanese, Italian, Norwegian], [Green,
Red, Yellow, Blue, White], [Painter, Diplomat, Violinist, Doctor, Sculptor],
[Dog, Zebra, Fox, Snail, Horse], [Juice, Water, Tea, Coffee, Milk]).
Zebra owner: japanese
Water drinker: norwegian
Englishman = Red, Red = Sculptor, Sculptor = Snail, Snail = Milk, Milk = 3,
Spaniard = White, White = Violinist, Violinist = Dog, Dog = Juice, Juice = 4,
Japanese = Green, Green = Painter, Painter = Zebra, Zebra = Coffee, Coffee = 5,
Italian = Blue, Blue = Doctor, Doctor = Horse, Horse = Tea, Tea = 2,
Norwegian = Yellow, Yellow = Diplomat, Diplomat = Fox, Fox = Water, Water = 1 ;
false.

?- 
```

**Problem 4:** Program the Sudoku Puzzle in CLF(FD). You should read input from the user which consists of a series of terms of the form:

$f(X,Y,Z)$ .

in a file called "input". The term  $f(X,Y,Z)$  states that the square at position  $(X, Y)$  has value  $Z$  ( $Z \in 1..9$ ). The input file is used to indicate the values given at the various squares in the puzzle. Your program should print the solution on the screen using write statements.

**Ans.**

```
sudoku(Rows) :-
    length(Rows, 9), maplist(same_length(Rows), Rows),
    append(Rows, Vs), Vs ins 1..9,
    maplist(all_distinct, Rows),
    transpose(Rows, Columns),
    maplist(all_distinct, Columns),
    Rows = [A,B,C,D,E,F,G,H,I],
    blocks(A, B, C), blocks(D, E, F), blocks(G, H, I).

blocks([], [], []).

blocks([A,B,C|Bs1], [D,E,F|Bs2], [G,H,I|Bs3]) :-
    all_distinct([A,B,C,D,E,F,G,H,I]),
    blocks(Bs1, Bs2, Bs3).

%----- PRINTING -----

writeRow([H1,H2,H3,H4,H5,H6,H7,H8,H9]) :-
    write('|'),
    %(var(H1) -> write('_'); write(H1)), write(' '),
    (var(H1) -> write('_'); write(H1)), write(' '),
    (var(H2) -> write('_'); write(H2)), write(' '),
    (var(H3) -> write('_'); write(H3)), write(' '), write('|'),
    (var(H4) -> write('_'); write(H4)), write(' '),
    (var(H5) -> write('_'); write(H5)), write(' '),
    (var(H6) -> write('_'); write(H6)), write(' '), write('|'),
    (var(H7) -> write('_'); write(H7)), write(' '),
    (var(H8) -> write('_'); write(H8)), write(' '),
    (var(H9) -> write('_'); write(H9)), write(' '),
    write('|'), nl.

%writeSudoku([]) :- nl.
writeSudoku([H1,H2,H3,H4,H5,H6,H7,H8,H9]) :-
    writeOuterLine([1,2,3,4,5,6,7,8,9]),
    writeRow(H1),
    writeRow(H2),
    writeRow(H3), writeInnerLine([1,2,3,4,5,6,7,8,9]),
    writeRow(H4),
    writeRow(H5),
    writeRow(H6), writeInnerLine([1,2,3,4,5,6,7,8,9]),
    writeRow(H7),
    writeRow(H8),
    writeRow(H9),
    writeOuterLine([1,2,3,4,5,6,7,8,9]), nl.
```

```

writeInnerLine([H|T]) :-
    write('|'),
    length([H|T],L), Temp is sqrt(L), N is ceil(Temp),
    Dashes is N+L*2-1, writeInnerDashes(Dashes).

writeInnerDashes(0) :- write('|'), nl.
writeInnerDashes(N) :- write('-'), N1 is N - 1, writeInnerDashes(N1).

writeOuterLine([H|T]) :-
    write('*'),
    length([H|T],L), Temp is sqrt(L), N is ceil(Temp),
    Dashes is N+L*2-1, writeOuterDashes(Dashes).

writeOuterDashes(0) :- write('*'), nl.
writeOuterDashes(N) :- write('-'), N1 is N - 1, writeOuterDashes(N1).

%----- 3 DIFFERENT PROBLEMS -----
:- dynamic problem/2.
% EASY
problem(1, [[_,_,_,_,_,_,_,_],
            [_,_,_,_,3,_,8,5],
            [_,_,1,_,2,_,_,_],
            [_,_,_,5,_,7,_,_],
            [_,_,4,_,_,_,1,_,_],
            [_,9,_,_,_,_,_,_],
            [5,_,_,_,_,_,7,3],
            [_,_,2,_,1,_,_,_],
            [_,_,_,4,_,_,_,9]]) .

% MEDIUM
problem(2, [[_,_,_,2,6,_,7,_,1],
            [6,8,_,_,7,_,_,9,_,_],
            [1,9,_,_,_,4,5,_,_,_],
            [8,2,_,_,1,_,_,_,4,_,_],
            [_,_,4,6,_,2,9,_,_,_],
            [_,5,_,_,_,3,_,2,8],
            [_,_,9,3,_,_,_,7,4],
            [_,4,_,_,5,_,_,3,6],
            [7,_,3,_,1,8,_,_,_]]) .

% HARD
problem(3, [[1,_,_,4,8,9,_,_,6],
            [7,3,_,_,_,_,_,4,_,_],
            [_,_,_,_,_,1,2,9,5],
            [_,_,7,1,2,_,6,_,_,_],
            [5,_,_,7,_,3,_,_,8],
            [_,_,6,_,9,5,7,_,_,_],
            [9,1,4,6,_,_,_,_,_,_],
            [_,2,_,_,_,_,_,3,7],
            [8,_,_,5,1,2,_,_,4]]) .

```

The screenshot shows a MATLAB script in the left pane and its execution output in the right pane. The script defines three Sudoku problems of increasing difficulty: EASY, MEDIUM, and HARD. Each problem is represented as a 9x9 matrix with some cells pre-filled. The script calls a function `problem` to solve each problem and `writeSudoku` to display the solution.

**Script Content (Left Pane):**

```

73 %----- 3 DIFFERENT PROBLEMS -----
74 :- dynamic problem/2.
75 % EASY
76 problem(1, [[_,_,_,_,_,_,_,_,_],
77             [_,_,_,_,_,_,_,_,_],
78             [_,_,_,_,_,_,_,_,_],
79             [_,_,_,_,_,_,_,_,_],
80             [_,_,_,_,_,_,_,_,_],
81             [_,_,_,_,_,_,_,_,_],
82             [_,_,_,_,_,_,_,_,_],
83             [_,_,_,_,_,_,_,_,_],
84             [_,_,_,_,_,_,_,_,_]]).
85 % MEDIUM
86 problem(2, [[_,_,_,_,_,_,_,_,_],
87             [_,_,_,_,_,_,_,_,_],
88             [_,_,_,_,_,_,_,_,_],
89             [_,_,_,_,_,_,_,_,_],
90             [_,_,_,_,_,_,_,_,_],
91             [_,_,_,_,_,_,_,_,_],
92             [_,_,_,_,_,_,_,_,_],
93             [_,_,_,_,_,_,_,_,_],
94             [_,_,_,_,_,_,_,_,_]]).
95 % HARD
96 problem(3, [[1,_,_,_,_,_,_,_,_],
97             [_,_,_,_,_,_,_,_,_],
98             [_,_,_,_,_,_,_,_,_],
99             [_,_,_,_,_,_,_,_,_],
100            [_,_,_,_,_,_,_,_,_],
101            [_,_,_,_,_,_,_,_,_],
102            [_,_,_,_,_,_,_,_,_],
103            [_,_,_,_,_,_,_,_,_],
104            [_,_,_,_,_,_,_,_,_]]).
105
106
107 %----- COMMANDS -----
108 %problem(1,Rows), sudoku(Rows), map
109 %problem(3,Rows), writeSudoku(Rows)
110 %

```

**Execution Output (Right Pane):**

The output shows the execution of the script for each problem. For each problem, it displays the solution matrix and the command used to solve it.

**Problem 1 (EASY) Solution:**

```

*-----*
| 1 2 3 4 5 6 7 8 9 |
| 4 5 6 7 8 9 1 2 3 |
| 7 8 9 1 2 3 4 5 6 |
| 2 3 4 5 6 7 8 9 1 |
| 5 6 7 8 9 1 2 3 4 |
| 8 9 1 2 3 4 5 6 7 |
| 3 4 5 6 7 8 9 1 2 |
| 6 7 8 9 1 2 3 4 5 |
| 9 1 2 3 4 5 6 7 8 |
*-----*
? - problem(1,Rows), writeSudoku(Rows), sudoku(Rows), writeSudoku(Rows).
true.

```

**Problem 2 (MEDIUM) Solution:**

```

*-----*
| 1 2 3 4 5 6 7 8 9 |
| 4 5 6 7 8 9 1 2 3 |
| 7 8 9 1 2 3 4 5 6 |
| 2 3 4 5 6 7 8 9 1 |
| 5 6 7 8 9 1 2 3 4 |
| 8 9 1 2 3 4 5 6 7 |
| 3 4 5 6 7 8 9 1 2 |
| 6 7 8 9 1 2 3 4 5 |
| 9 1 2 3 4 5 6 7 8 |
*-----*
? - problem(2,Rows), writeSudoku(Rows), sudoku(Rows), writeSudoku(Rows).
true.

```

**Problem 3 (HARD) Solution:**

```

*-----*
| 1 2 3 4 5 6 7 8 9 |
| 4 5 6 7 8 9 1 2 3 |
| 7 8 9 1 2 3 4 5 6 |
| 2 3 4 5 6 7 8 9 1 |
| 5 6 7 8 9 1 2 3 4 |
| 8 9 1 2 3 4 5 6 7 |
| 3 4 5 6 7 8 9 1 2 |
| 6 7 8 9 1 2 3 4 5 |
| 9 1 2 3 4 5 6 7 8 |
*-----*
? - problem(3,Rows), writeSudoku(Rows), sudoku(Rows), writeSudoku(Rows).
true.

```

**Note:**

\* All **source files** for each problem 'i' is saved as **hw5qi.lp**. Please, refer them for evaluation.

\* All the **screen shots** in this document are also attached as **a5qi.png**.