Madhuri Pyreddy
9 November 2019

CSCI 117 Lab 11

1. N-queens tells us that the first argument gives us a dimensional board for the number of queens. The first query, “ n_queens(8, Qs), label(Qs)”, for which label1 returns values. The first argument is a positive integer and the second argument is a list of length of the first argument. In the query, 8 is the 8*8 board.The first two full loops of safe_queens([Q|Qs], Q0, D0) affect the constraints of column values for the queens Q2...Q7 (assuming 8 queens, and Q0 = 1, Q1 = 5) because they show in a 8*8 board where each queen is assigned to each of the columns since it is impossible to place two queens in the same column. We need to find only which row each of the queens are placed. In Q0=1, the first queen is placed at the first square of row 0. In Q1 = 5, the second queen is placed at the fifth square of row 1. When we run two loops, they show the different possible assignments the queens are placed in the search space from Q2 to Q7. The 1st loop is (Q2=8, Q3=6, Q4=3, Q5=7, Q6=2, Q7=4) and the second loop is (Q2=4, Q3=6, Q4=8, Q5=2, Q6=5, Q7=3). Constraints are determined by how the variables and set of values are chosen so when the queens are placed in different rows, they can attack each other depending on different locations.
2. In the sudoku puzzle, we included the library CLPFD in line 4. In line 8, the function sudoku(Rows) takes a list of list as arguments. In line 9, the length takes in 9 rows and maps The function append inserts variables with range of possible values into the lists. In Vs, case 1 and 9 are ensured. With maplist, we call all_distinct on all lists inside the list from the library. We transpose rows into columns, which is part of the library. All_distinct makes sure that every value in the list just occurs once, as forced by the rules of Sudoku. Transpose rows and columns in Sudoku. We call all_district again to make sure all numbers just occur once in the columns.Convert Rows in n*n block to check whether they are all self-distinct blocks. Again, we use maplist to call predicate label1 to make sure all domains have one concrete value. We express that the predicate blocks is true when it gets three empty lists as inputs. If blocks/3 has non-empty lists, Prolog will use the pipe (|) to split up given rows into the first three rows and the rest(tail) to use later. We will recursively call the list until they are split to empty lists.
3. % You are on an island where every inhabitant is either a knight or a

    % knave. Knights always tell the truth, and knaves always lie. The
    % following examples show how various cases can be solved with CLP(B),
    % Constraint Logic Programming over Boolean variables.

    % These examples appear in Raymond Smullyan's "What Is the Name of
    % this Book" and Maurice Kraitchik's "Mathematical Recreations".

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


% We use Boolean variables A, B and C to represent the inhabitants.
% Each variable is true iff the respective inhabitant is a knight.
% Notice that no search is required for any of these examples.


% Example 1: You meet two inhabitants, A and B.
%           A says: "Either I am a knave or B is a knight."

example_knights(1, [A,B]) :-
        sat(A=:=(~A + B)).


% Example 2: A says: "I am a knave, but B isn't."

example_knights(2, [A,B]) :-
        sat(A=:=(~A * B)).


% Example 3: A says: "At least one of us is a knave."

example_knights(3, [A,B]) :-
        sat(A=:=card([1,2],[~A,~B])).


% Example 4: You meet 3 inhabitants. A says: "All of us are knaves."
%           B says: "Exactly one of us is a knight."

example_knights(4, Ks) :-
        Ks = [A,B,C],
        sat(A=:=(~A * ~B * ~C)),
        sat(B=:=card([1],Ks)).


% Example 5: A says: "B is a knave."
%           B says: "A and C are of the same kind."
```

```
% What is C?

example_knights(5, [A,B,C]) :-
        sat(A=:= ~B),
        sat(B=:=(A=:=C)).


% Example 6: B says: "I am a knight, but A isn't."
% A says: "At least one of us is a knave."
example_knights(6, [A,B,C]) :-
  sat(B=:=(~B + A)).
  sat(B=:= ~A),
  sat(B=:=(A=:=C)).
```

Output: **A** = B, **B** = 1.

```
% Example 7: A says: "At least one of us are knights."
% B says: "Either I am a knight or A is a knave."

example_knights(7, Ks) :-
      Ks = [A,B,C],
 sat(A=:=(A * B * C)),
 sat(B=:=(~B + A)).
```

Output: **Ks** = [1, 1, 1].

```
% Example 8: B says: "All of us are knights."
% A says. "Exactly one of us is a knave."

example_knights(8, Ks) :-
        Ks = [A,B,C],
        sat(B=:=(~A * ~B * ~C)),
        sat(A=:=card([1],Ks)).
```

Output: **Ks** = [1, 0, 0].

4.

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

reverse([],[]).  %reverse of empty is empty - base case
```

```
reverse([H|T], RevList):-
   reverse(T, RevT), append(RevT, [H], RevList).  %concatenation

helpNeeded([],[],C,[T3]):- T3 #= C.
helpNeeded([H1|T1],[H2|T2],Carry2,[H3|T3]):-
   H3 #= (H1+H2+Carry2) mod 10,
   Carry #= (H1+H2+Carry2) div 10,
   helpNeeded(T1,T2,Carry,T3).

crypt1([H1|L1],[H2|L2],[H3|L3],L4) :-
   L4 ins 0..9,
   H1 #\= 0, H2 #\= 0,
   all_different(L4),
   reverse([H1|L1], Out1), reverse([H2|L2], Out2), reverse([H3|L3], Out3),
   helpNeeded(Out1, Out2,0, Out3).
```

?- *crypt1*([A,N,G,E,L],[B,A,G,E,L],[L,L,C,B,N,E],[A,N,G,E,L,B,C]),
labeling([ff],[A,N,G,E,L,B,C]).

Output:
A = 5,
B = 6,
C = 9,
E = 2,
G = 3,
L = 1,
N = 4

?-*crypt1*([C,A,R,R,O,T],[P,A,R,R,O,T],[T,T,D,P,P,A,O],[C,A,R,O,T,P]),
labeling([ff],[C,A,R,O,T,P]).

Output:
A = 5,
B = 6,
C = 9,
E = 2,
G = 3,
L = 1,
N = 4

?- *crypt1*([J,A,Z,Z,E,D],[B,U,Z,Z,E,D],[D,Z,Y,J,J,A,E],[J,A,Z,E,D,B,U]),
labeling([ff],[J,A,Z,E,D,B,U]).

Output:
A = Y, **Y** = 4,
B = 7,
D = 1,
E = 2,
J = 6,
U = 0,
Z = 3