Javier Escareno
April 12th 2024

CSCI 117 LAB 10

1)The safe_queens function in the context of the N-Queens problem generates constraints dynamically to ensure safe placement of queens on an N x N chessboard. For example if N = 4 ,we start by initializing a list of 4 unbound variables. As we execute safe_queens(Qs, 1, []), each variable represents a potential row position for a queen in its respective column (Q0). The function adds constraints to each variable to enforce that no two queens share the same column or threaten each other diagonally. Initially, each variable is constrained to values specified. As queens are placed, diagonal constraints (D0) are updated to prevent conflicts with previously placed queens. This process continues until all queens are positioned, resulting in a valid arrangement (Qs) satisfying all constraints. The third argument (D0) plays a crucial role in tracking and updating diagonal constraints throughout the placement process, ensuring the solution adheres to the rules of the N-Queens puzzle for boards of any size.

2.) Sudoku

```
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).

problem(1, [[_,_,_, _,_,_, _,_,_],
            [_,_,_, _,_,3, _,8,5],
            [_,_,1, _,2,_, _,_,_],

            [_,_,_, 5,_,7, _,_,_],
            [_,_,4, _,_,_, 1,_,_],
            [_,9,_, _,_,_, _,_,_],

            [5,_,_, _,_,_, _,7,3],
            [_,_,2, _,1,_, _,_,_],
            [_,_,_, _,4,_, _,_,9]]).
```

1) With the 1st line the length is defined to create a 9x9 square for the puzzle “board”.
2) The appends then inserts variables within the range of possible values into the list. It can be seen that cases 1..9 are included.
3) Using maplist we also call allDistinct to ensure the numbers imputed are all different from one another with non repetition from the list given for the row.
4) We then transpose the rows and columns of the puzzle.
5) Using maplist we also call allDistinct to ensure the numbers imputed are all different from one another with non repetition from the list given for the column.
6) For rows it will convert Rows in the NxN block to check if they are in their own block.
7) The blocks are where the variables are contained, and each domain has only one concrete value. Moreover, the predicate block is true if it receives three empty lists as its input. If the blocks/3 is non-empty, Prolog will use the pipe to split up the rows and store the rest for later.
8) The values are then inserted into the 9 by 9 table for the sudoku puzzle following all the restrictions set by the program.

3.) Knight and Knaves

Example 1: 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=:=(C=:=A)).
```

```
Output: A = B, B = 1
```

Example 2: B says: “Either one of us is a knight.”

```
example_knights(7, [A,B]) :-

     sat(B=:=card([2,3],[~A,~B])).
```

Output: A = 1, B = 0

Example 3: C says: “We are all knaves.”

A says: “Only one of us is a knight.”

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

Output: Ks = [1,0,0]

Part 2:

```
:- 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,L,E],[T,A,B,L,E],[L,L,C,B,N,E],[A,N,G,L,E,B,C]), labeling([ff],[A,N,G,L,E,B,C]).
```

```
A = 1,

B = 3,

C = 4,

E = 0,

G = 9,

L = 6,

N = 2,

T = 65
```