MA410 Artificial Intelligence Practical

Sudoku Problem - Writing The Dreaded Prolog Program

To create Sudoku boards from scratch

1. store a list keeping account of all possible positions and tag to each symbol.
2. start with the first symbol.
3. start with column 1
4. choose a valid position in the column for the selected symbol. Record this fact.
5. for each symbol's list, delete the position chosen.
6. (only) in selected symbol's list, delete all positions attacked by this position.
7. now choose a position in next column and follow steps 4-6. Backtrack if necessary.
8. repeat step 7 until we have placed a symbol in all the columns.
9. Now repeat steps 3-7 for each of the symbols.

To solve a Sudoku problem

First we point out the difference from above:

- We are given a list of initial positions say `IP`.

How to deal with this:

- Delete all values in `IP` from our original list that keeps all possible positions.
- Add squares in `IP` to our list of occupied squares.
- Delete all attacked squares due to `IP`.
- In place of step 3 above, we need to add in the condition that if there is a value in `IP` corresponding to the symbol and column chosen, we skip this and move to the next column to try and place symbol.

Example

Let's take an example of a 2×2 board. This makes no sense for real Sudoku but we'll use it for illustrative purposes and say attacks only happen on rows and columns). (Note you may need to alter definitions of predicates so they are more generalised.)

Assume we have created predicates:

`removeAll(L,B,C)`: removes a list `L` of items from `B` to make `C`.
`attack([X1,Y1],[X2,Y2])`: returns true if position `[X1,Y1]` attacks position `[X2,Y2]`

Step 1: We create a list in prolog, say:

```
L = [[a,1,1],[a,1,2],[a,2,1],[a,2,2], [b,1,1],[b,1,2],[b,2,1],[b,2,2]].
```

The letters `a` and `b` refer to the symbols used[1]. We'll also have an empty list `LTaken`, that will store the positions we've chosen. We have an association made by storing the symbol name along with the position in a list.

2: We choose the first symbol `a`.
3: We choose column `1`.
4: We choose a valid position say `[1,1]` for `a`.
5: We remove each instance of `[1,1]` for every symbol as follows:

```
findall([X,1,1],member([X,1,1],L),L11).  %Create a list of all instances
removeAll(L11, L, LNew).
```

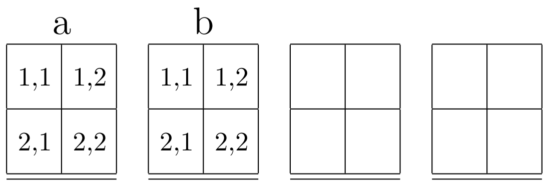
Note we now have a new list `LNew` that we should refer to

```
LNew = [[a,1,2],[a,2,1],[a,2,2], [b,1,2],[b,2,1],[b,2,2]].
```

6: Delete positions attacked by `[1,1]`:

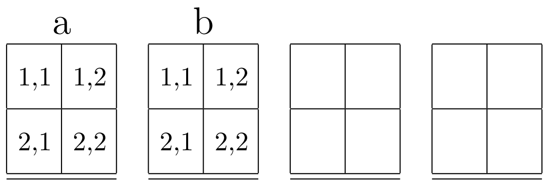
```
findall([a,X,Y], attack([1,1],[X,Y]), LAttacks).  %List of attacked squares
removeAll(LAttacks, LNew, LNew2).
```
(so `LNew2 = [[a,2,2], [b,1,2],[b,2,1],[b,2,2]].`)

7: We go to next column and choose the only position available `[2,2]`.
Keep record of values `LTaken`, `LNew` and `LNew2` and follow steps to create the board:



Record of List Values

This time have initial values chosen - just like solving a sudoku.



Record of List Values

[1]in Sudoku, numbers are generally used but trying to avoid confusion here as regards clash in notation

Prolog Tasks

1. Edit `sud.pl`.
2. Write predicates:
   (a) `listNums(N1,N2,L)`: L= [N1,...,N2].
   (b) `boardPos([R,C],N)`: `[R,C]` is a board position on an N × N board.
   (c) `symbolBoardPos([S,R,C],N)`: `[S,R,C]` = symbol with board position on N × N board.
   (d) `allBoardPos(N,L)`: L=`[[1,1],....,[N,N]]`.
   (e) `allSymbolBoardPos(N,L)`: L=`[[1,1,1],....,[N,N,N]]`.
3. Write predicates:

	returns true if *P1=[R1,C1]* & *P2=[R2,C2]*
`on_same_col([R1,C1],[R2,C2])`	are on same column.
`on_same_row([R1,C1],[R2,C2])`	are on same row.
`in_same_subsquare([R1,C1],[R2,C2],N)`	are in same subsquare on N × N board.
`attack(P1,P2,N)`	attack each other (subsquares on N × N board).
`attack_list_restricted(S,L,P1,P2,N)`	as previous but `[S|P1]`, `[S|P2]` ∈ `L` for some symbol `S`.

4. (a) Create predicates `deleteAttacksFromList`, `deleteIntersectingPosFromLists`, `trySymbolInCol`, `trySymbolsAndColsInTurn`. *(You need to figure out what parameters and the number of them to use.)*
   (b) Write out ideas first and create predicate `createSudoku(N,L)` where L is a list of possible positions on an N × N board in the form `[[S1,R1,C1],[S2,R2,C2], ....]`.
5. (a) Create predicate `removeInitialPositions`.
   (b) Create predicate `solveSudoku(N,IP,L)` where `L` is a list which solves the N × N sudoku with initial board positions `IP` in form `[[S1,R1,C1],[S2,R2,C2], ....]`.
6. Create predicate `solveSudokuAndPrint(N,IP)` that uses `print_sudoku(L,N)` (as in `sudpr.pl`).
7. In addition to earlier predicates, use `createIPList` (in `sudipfns.pl`) in order to create predicate `sudoku(Board)` that prints out the solution to the sudoku board `Board`, where `Board` is written in the usual Sudoku format, e.g.

```
[[_,_,_,_,_,_,_,_,_],
 [_,_,_,_,_,3,_,8,5],
 [_,_,1,_,2,_,_,_,_],
 [_,_,_,5,_,7,_,_,_],
 [_,_,4,_,_,_,1,_,_],
 [_,9,_,_,_,_,_,_,_],
 [5,_,_,_,_,_,_,7,3],
 [_,_,2,_,1,_,_,_,_],
 [_,_,_,_,4,_,_,_,9]]
```

8. Use examples from `test_suds.pl` to test the predicate `sudoku(Board)`.