CSCE 580: Artificial Intelligence
Coding Homework 3: Prolog
Due: 2/23/2024 at 11:58pm

In this homework, you will implement Prolog predicates that describe different scenarios in tic-tac-toe. You can invent as many predicates as you need to solve the problem.

Your code must run in order to receive credit.

Installation

Download swi-Prolog from https://www.swi-prolog.org/download/stable.

The entire GitHub repository should be downloaded again as changes were made to other files. You can download it with the green "Code" button and click "Download ZIP".

Helper Predicates

`seq(State, [Elem1, Elem2, Elem3], [Pos1,Pos2,Pos3])` is true if, for a state, the three elements (which will be consecutive elements in a row, column, or diagonal) are in the corresponding positions. The elements can be "x", "o", or "b" (blank spaces are denoted with a "b".). The positions can be integers 0 through 8 and correspond to the 9 positions on a tic-tac-toe board.

Make sure to put the `valid_state(S)` as the first predicate in your definition of the predicates for your homework. This will ensure that the solutions found are valid states.

You can use the `print_state(S)` predicate to print a 3x3 grid to the terminal every time you find a solution. You can do this by adding it on to the end of the predicate you are defining to. For example `two_ways_x(S) :- valid_state(S), pred1, pred2, ..., print_state(S).`

Two Ways for X to Win (100 pts)

Define a predicate `two_ways_x(S)` that is true for state `S` when there are at least two different ways that X can immediately win (get three in a row). In English, the predicate should be true if there are two rows, columns or diagonals that both have two X's and one blank and the positions of the blanks in the two sequences are different. Note that, though the blanks must be in different positions, the X's can overlap.

While you can invent as many predicates as you need, the final predicate that depends on them to solve the problem must be named two_ways_x(S).

A subset of the solutions is shown below. Depending on your implementation, your solutions may be returned in a different order and there may be duplicates.

First, load your Prolog file with `swipl coding_hw/coding_hw3.pl`

```
?- two_ways_x(S).
oob
obx
bxx
S = [o, o, b, o, b, x, b, x, x] ;
oob
obx
bxx
S = [o, o, b, o, b, x, b, x, x] ;
oob
obx
xbx
S = [o, o, b, o, b, x, x, b, x] ;
oob
obx
xbx
S = [o, o, b, o, b, x, x, b, x] ;
oob
oxb
xbx
S = [o, o, b, o, x, b, x, b, x] ;
oob
oxb
xbx
S = [o, o, b, o, x, b, x, b, x] ;
oob
oxb
xxb
S = [o, o, b, o, x, b, x, x, b] ;
oob
oxb
xxb
S = [o, o, b, o, x, b, x, x, b] ;
oob
box
bxx
S = [o, o, b, b, o, x, b, x, x] ;
oob
box
bxx
S = [o, o, b, b, o, x, b, x, x] ;
oob
box
xbx
S = [o, o, b, b, o, x, x, b, x] ;
```

Extra Credit (20 pts)

Define a predicate `no_ways_x(S)` that is true for state `S` when there is no way that X can win (get three in a row).

While you can invent as many predicates as you need, the final predicate that depends on them to solve the problem must be named no_ways_x(S).

What to Turn In

Turn in your implementation of `coding_hw/coding_hw3.pl` to Blackboard.