Laboratory: Prolog (3)

Summary: We conclude our exploration of the Prolog programming language by studying Tate’s puzzle-solving programs.

Prerequisites: The first Prolog lab. The second Prolog lab. Tate, Section 4.4.

Contents:

- Preparation
- Exercises
  - Exercise 1: Improving Output in Sudoku
  - Exercise 2: Testing Sudoku
  - Exercise 3: Improving Tate’s Solution
  - Exercise 3: Extracting “Squares”
  - Exercise 4: Improving the Sudoku UI
  - Exercise 5: Fun with Eight Queens
  - Exercise 6: Watching Computation
  - Exercise 7: The N-Queens Problem
- For Those with Extra Time
  - Extra 1: Real Sudoku
  - Extra 2: A Knight Problem

Preparation

a. Create a directory for the lab.

b. Make copies of Tate’s Sudoku example and Eight queens example.

Exercises

Exercise 1: Improving Output in Sudoku

a. Verify that Tate’s Sudoku program works as advertised on the sample input.

b. As many of you noted, the output of the program is not particularly nice. Write a print_sol(Solution, Cols, Rows) predicate that prints out the solution appropriately.

You can use the write and print predicates to print values. (You may find it useful to print the value ‘.’.) The nl predicate prints a newline.
Exercise 2: Testing Sudoku

Tate shows us only one example of his Sudoku solve in action. Try the following:

a. Find all solutions to the sample input.

b. Try one or two examples of your own devising.

c. Verify that the predicate fails on an unsolvable puzzle.

Exercise 3: Improving Tate’s Solution

I’m not sure about you, but I cringe a bit at the hard-coded naming of rows and columns in Tate’s solution. (I expect that they’re there for clarity, rather than elegance.)

a. Write a predicate, `rows(Puzzle, NumCols, NumRows, ListOfRows)`, that builds a list of the rows in the puzzle. (After executing the predicate, `ListOfRows` should contain the equivalent of `[Row1, Row2, Row3, Row4]`.)

b. Write a predicate, `cols(Puzzle, NumCols, NumRows, ListOfCols)`, that builds a list of the columns in the puzzle.

c. Rewrite Tate’s solution to use these two predicates.

Exercise 3: Extracting “Squares”

a. Write a predicate that lets you extract a list of “squares” from a Sudoku puzzle. (You may want to take the size of the squares as a parameter.)

b. Rewrite Tate’s solution to use this new predicate.

Exercise 4: Improving the Sudoku UI

As you may have noted, Tate requires you to specify the whole Sudoku board using either variables or constants. That seems excessive. You can use the `length(List, Len)` predicate to build a list of a particular length.

a. Write a predicate, `value_at(Puzzle, Cols, Rows, Col, Row, Val)`, that holds when the value at column `Col` and row `Row` is `Val`.

b. Write a predicate, `puzzle(Cols, Rows, Values, Puzzle)` that holds if `Puzzle` is a `ColsxRows` Sudoku puzzle with values as specified by `Values`, which is a list of `(Col, Row, Value)` triplets.
**Exercise 5: Fun with Eight Queens**

a. Verify that Tate’s eight-queens algorithm works as advertised.

b. Determine if there is a solution to the eight-queens problem with a queen at (4,5).

c. Pick some queen position in which there is a solution, and determine how many solutions there are with a queen at that position.

**Exercise 6: Watching Computation**

Extend Tate’s solution so that you can see it try each arrangement.

**Exercise 7: The N-Queens Problem**

While the eight queens problem is solvable, not all N-queens problems are solvable. For example, it is not possible to put 3 queens on a 3x3 board in such a way that no queen can take any other queen.

Rewrite Tate’s solution so that you can use it to help you determine for what $N \leq 9$ the N-queens problem is solvable.

**For Those with Extra Time**

Choose one or more of these problems.

**Extra 1: Real Sudoku**

Using the procedures you wrote earlier, build a solver for standard Sudoku problems (27x27 in 3x3 grids).

**Extra 2: A Knight Problem**

Write a Prolog program that can help you figure out how many knights you can put on an 8x8 chess board in such a way that no two knights can take each other. (You may find it easiest to write a predicate that checks if you can place $N$ knights on the board.)