Instructions

Instructions. We will be starting Prolog, which is a declarative language. While SML was more declarative than some of the other languages you are familiar with (e.g. Python and Java), Prolog is even more so. This means that you need to start thinking in terms of descriptions of solutions to problems rather than instructions on how to solve a problem (which is an imperative thing). This assignment is to get you thinking about what this looks like. If you can embrace the declarative nature of Prolog, you'll find that it is a very powerful language that can do some interesting things in just a few lines of code.

Question 1

> Sudoku is a puzzle involving a 9X9 grid that must be filled with the numbers 1 through 9.
>
> For this question, I simply want you to describe what a Sudoku solution would look like given the input of a 9X9 grid.
>
> Clearly, the grid must be filled out with 81 numbers, what properties does that grid have to have?
>
> It's helpful if you think of these properties in terms of what has to be true about each row, column, and 3X3 subgrid. Or, alternatively, you can think in terms of what need to be true about the placements for each of the numbers 1 through 9. There will be 9 1's, 9 2's, 9 3's, etc., but what has to be true about the placement of those 9 1's in the final grid?

Let M represent a generic valid Sudoku solution grid (matrix) with entries m_{ij}.

We declare M as having the following properties:

$$M \in \{x \in \mathbb{Z} \mid 1 \leq x \leq 9\}^{9\times 9}$$

$$\forall i \in \{1, 2, \dots, 9\}, |\{m_{ij} \mid j \in \{1, 2, \dots, 9\}\}| = 9$$

$$\forall j \in \{1, 2, \dots, 9\}, |\{m_{ij} \mid i \in \{1, 2, \dots, 9\}\}| = 9$$

$$\forall p, q \in \{0, 1, 2\}, |\{m_{ij} \mid i \in \{3p{+}1, 3p{+}2, 3p{+}3\}, j \in \{3q{+}1, 3q{+}2, 3q{+}3\}\}| = 9$$

The set of all valid Sudoku solutions S is defined as:

$$S = \{M \mid M \text{ satisfies the above properties}\}$$

Question 2

> One of the things Prolog does well is find solutions to difficult problems. Sudoku is a difficult problem for a computer to solve (difficult as in "time-consuming"), and in some forms can be considered NP-Complete. Another difficult problem for computer to solve is known as the Knapsack Problem, which has multiple variations. The 0-1 Knapsack Problem is also NP-Complete. The problem description for the 0-1 Knapsack is that you have a knapsack with a weight capacity of C. You also have a set of items, each with a value v and a weight w. The goal is to choose a subset of those items so that the total weight does not exceed C and so that the total value is maximized. The "0-1" part of this problem indicates that there are no duplicate items – each item can either be included in the knapsack (1) or not (0). Finally, to simplify the problem a little, we can add another parameter, which is a value threshold V. So instead of requiring a maximal value, we are simply looking for a subset of items whose total weight does not exceed C and whose total value is at least V. Your task here is to describe a solution to this problem as clearly and succinctly as possible.

Let C represent the weight capacity of the knapsack and V represent the value threshold. Let I be the set of items, where each item $i \in I$ has a value v_i and a weight w_i.

Let set K represent a generic valid solution to this variation of the 0-1 Knapsack Problem.

We declare K as having the following properties:

$$K \subseteq I$$

$$\sum_{i \in K} w_i \leq C$$

$$\sum_{i \in K} v_i \geq V$$

The set of all valid solutions S to this variation of the 0-1 Knapsack Problem is defined as:

$$S = \{K \mid K \text{ satisfies the above properties}\}$$