Project Details

In this project, you will write a simple program in Prolog to solve the following puzzle:

In a children's story, three foxes and three hens are traveling together. They come to a river they must cross. There is a boat, but it will hold no more than two of them at a time, so at least six trips in each direction will be required (one animal has to row the boat back each time). The problem is that the hens cannot trust the foxes; if in the process of crossing the river we ever end up with more foxes than hens on either shore, the foxes are likely to eat the hens. We need a series of moves (trips across the river in the boat) that will get the entire party across safely.

Your main predicate should be a goal `solve(P)`. It should instantiate `P` to be a list of configurations, each of which indicates the location of the animals and the boat. If you request additional solutions (by typing a semicolon) the interpreter should give you additional lists, as many as there are unique solutions. One simple, minimal way to represent a configuration is to give the number of foxes, hens, and boats on the near (left) shore: $\langle LF, LH, LB \rangle$, where $0 \leq LF \leq 3$, $0 \leq LH \leq 3$, and $0 \leq LB \leq 1$.

You can think of the state space of this problem as a graph in which nodes are configurations and edges represent feasible boat trips. It's a large graph, however, and you won't want (nor in fact are you permitted) to represent it explicitly in your Prolog database. From any given state you'll need to figure out the set of possible neighbors to explore. You'll also want to keep track of where you have been, so you don't get stuck in a cycle.

You may find it helpful to use the `not` operator. You may also want to use the `cut` to avoid unnecessary backtracking. You are not permitted to use `assert`, `retract`, or other database-modifying predicates. (For the record, you must **compute** your solutions. You are not allowed to figure them out by hand and simply write a program that prints them.)

Division of labor and writeup

For this project, you will work in pair or individually. Either way, your README should note if you work alone or you work with a partner. If you work in pair, clearly specify each person's contribution.

Grading scale

The project will be graded based on your source code and the README file:

- Code:
  - Correctness, completeness and efficiency (70%): Your code should implement everything that was required in the assignment. It should produce the right output given normal, expected inputs, and some sort of reasonable response to unexpected inputs. Unless otherwise instructed, and as long as it does not severely compromise programming style, you are expected to use the most efficient data structures and algorithms.
  - Programming style (including internal documentation and program organization) (15%): Your code should have appropriate abstractions, data structures, algorithms, and variable names; declarations for all constants; and a judicious number of helpful comments. You should think of programming as explaining to the readers of your programs what you want the computer to do.
- README file:

- Completeness (10%): Your write-up should include all the items discussed in the README files.
- Readability (5%): Your write-up should be well organized, clear, and concisely presented.