Bill Qualls – CSC480 – Assignment 3 – Strips Report

This was a great project! I found the two previous exercises to be frustrating as the solution was too tightly coupled to the problem. But that all changed with this project. With **STRIPS** we have a general purpose problem solving tool.

The class `Solver` is what actually implements the **STRIPS** solver. It is a surprisingly short class. It takes a start state (instance of `State`), goal state (instance of State), and an array of defined actions (each of which is an instance of `Action`).

A `State` consist of arrays of `Proposition`. Each `Proposition` consists of a name, an array of objects, and (optionally) a boolean. For example "disk", { "1" }, or "onDisk, { disk1, disk2 }".

An `Action` actually consists of **precondition** list, an **add list**, and a **delete list**. Each of these lists is implemented as a `State`, which as mentions, is an array of `Proposition`. When the `Proposition` is a part of a precondition, the boolean indicates if the Proposition is allowed: in this way I extended the **STRIPS** model to include negation. (For example, in the Tower of Hanoi puzzle, one condition is that the disk you are attempting to move is NOT the same disk you moved before. Keeping track of the prior move is not a task which belongs in the `Solver` and it is relegated to those classes which extend `Action` and override its doit() method.)

The search tree consist of `Node`s. A `Node` consists of an `Action` and the `State` after that `Action` is implemented (the "new" current state). A `Node` also consists of a pointer to its parent, so I can trace back to the top of the tree and thereby recall all actions taken. Finally, a `Node` also consists of a level number (how deep into the tree is this node), an ID (a sequentially assigned number), both of which proved interesting and useful in developing the program so I could see how the tree was growing.

Each problem class consists of defining **objects** (for example: disk1, blockA), **actions** (for example: moveTopDiskToEmptyPeg, moveBlockFromBlockToTable), the **starting state** (for example: disk4IsOnLeftPeg, blockBIsOnTable), and **goal state** (for example: disk4IsOnMiddlePeg, blockBIsOnBlockC). The problem then calls the `Solver`, and the `Solver` solves the problem (finds the `Action`s necessary to go from the starting `State` to the goal `State`), then passes a list of `Action`s taken back to the calling problem.

I used **STRIPS** to solve several problems, each of which may be accessed through the provided menu program.

(BQ – 02/22/2014)