# Homework 1: Chapters 1 - 5

The following exercises are due at the beginning of class on **Tuesday, October 4**. Some of these problems may take a while to solve, so I recommend that you work on this assignment over the course of multiple days.

1. ***[10 pts.]*** Prove that state-based agents are equivalent in expressive power to standard agents, i.e. that for every state-based agent there is a behaviorally equivalent standard agent and vice versa.
2. ***[25 pts. total]*** Consider the vacuum-world example from Chapter 3 (pp. 52-54). Note: This is different from the environment we used for our programming assignment. In particular, there are no obstacles and the agent does not need to return home after it is done cleaning the room.
   a) ***[10 pts.]*** Give the full definition (using pseudo-code if desired) of the *new* function, which defines the predicates to add to the agent's database. For convenience, you may assume that agent stores a global variable *action* which contains the last action it executed.
   b) ***[15 pts.]*** Use the book's version of the subsumption architecture approach to design a purely reactive agent for this environment. You can assume a "wall" percept which tells the agent that it is facing the boundary of the room. You do not need to implement this agent, only give a formal description of it. How does it compare with the logic-based example? Which do you think will perform better? Which is simpler?
3. ***[15 pts.]*** Consider the Mars explorer example from Chapter 5 (pp. 87-90). Assume that there are no radioactive crumbs, but agents are able to broadcast messages to each other. Outline how one would design an InterRRaP agent for this problem. In particular, you must discuss how each of the three layers (behavior, plan, and cooperation) contribute to decision making and what kind of information is stored in the three knowledge bases (world model, planning knowledge, social knowledge).
4. ***[40 pts. total]*** Now consider a single-agent variation of the Mars explorer example. Assume that the world is specified as a grid with the mothership in the center at location(0,0). The agent has actions: **GoNorth, GoEast, GoSouth, GoWest, PickupSample,** and **DropSample** and the percepts: **Obstacle(*direction*),** **Gradient(*direction*)**, **CarryingSample** and **AtSample.**
   a) ***[10 pts.]*** Formulate the operations available to the agent using the STRIPS notation. Note, you must decide what set of predicates are sufficient. Hint: You may augment your STRIPS notation with the basic arithmetic operators.
   b) ***[15 pts.]*** Now design a PRS-style plan library for this problem. Remember, an important aspect of PRS plans is that they can contain goals as well as concrete actions. Then can also include conditionals and loops. As long as the components of your plan are unambiguous, you do not have to use a specific syntax, but you may want to refer to the Jam example (Fig. 4.5, p. 82) for ideas. Which ideas if/any were difficult to describe with plans? Did you have to extend the notation or assume additional functionality in order to achieve a satisfactory design?

c) ***[15 pts.]*** Assuming rock samples tend to appear in clusters, think about designing a BDI agent to solve the problem. What types of knowledge will be needed to form its Beliefs? Summarize what desires should be generated by the options() function, and under what conditions each is possible. Finally, describe how the filter() function should select between competing options. You don't have to give pseudo-code, but you should give enough details so that I could see how to flesh out your design.

5. ***[10 pts.]*** Consider the following precondition and effect axioms. Assuming that these are the only axioms that are relevant to the fluent ***closed***, write a successor state axiom for it.

   Poss(Open(d),s) ⇔ closed(d,s) ∧ nextTo(d,s) ∧ ¬locked(d,s)
   Poss(Close(d),s) ⇔ ¬closed(d,s) ∧ nextTo(d,s)
   Poss(Explode(e),s) ⇔ combustible(e,s)
   Poss(Open(d),s) ⇒ ¬closed(d, Do(Open(b),s))
   Poss(Explode(e),s) ∧ near(e,d,s) ⇒ ¬closed(d, Do(Explode(e),s))
   Poss(Close(d),s) ⇒ closed(d, Do(Close(b),s))