CS 227: Homework #4
Spring 2011

Assigned: Tuesday, May 24th
Due Dates: June 2nd

Notes:

You may work on this homework in groups of two. If you work in a group, only one of you needs to submit your report. You should submit your report for problem2 separate from your report for the rest of the problems to enable faster grading. Both members of the group will receive the same grade for the report.

Problem 1: Qualitative Representation and Reasoning

Part (a): Formalizing qualitative relations

We considered the relations that represent direct influence (I+ and I-), and proportionality (P+ and P-). Write the meaning of these four relations in first order logic.

Part (b): Representing qualitative knowledge

Consider the following paragraph of knowledge

> *A saturated fatty acid has only single bonds between its carbon atoms, so its molecule is flat. This means multiple molecules pack tightly, so its surface area per unit mass is low. An unsaturated fatty acid has at least one double bond between its carbon atoms, so it has a kinked structure. This means that multiple molecules pack loosely, so its surface area per unit mass is high. Mechanical digestion involves physically separating molecules. Molecules that pack loosely, with high surface area per unit mass, are easier to separate. Therefore, unsaturated fatty acids are easier to mechanically digest.*

Identify the quantities that are mentioned in this paragraph, and specify which of the four qualitative relationships from part(a) hold amongst those quantities.

Part (c): Reasoning with qualitative knowledge

Consider the following question:

Which is easier to break down through mechanical digestion: saturated fatty acid, or unsaturated fatty acid?

By listing the reasoning steps, show how the qualitative relationships defined in part (a) can be used to answer the above question?

Problem 2: Classical Planning

In this problem, we will experiment with the transport puzzle Sokoban. You can find a description of the puzzle at http://en.wikipedia.org/wiki/Sokoban. The goal is to encode three different configurations of the Sokoban of varying sizes using the Planning domain definition language (PDDL), and then use the off-the-shelf planner FF, and plot its performance.

a. You are welcome to create your own configuration of the Sokoban puzzle. You are also welcome to choose from pre-defined puzzles available at http://webdocs.cs.ualberta.ca/~games/Sokoban/status.html. To see example PDDL descriptions for the game of Iceblox, refer to the paper http://www.stanford.edu/class/cs227/Readings/PDDLPlanningForArcadeGamePlaying.pdf Additional examples of sample PDDL are available at http://planning.cis.strath.ac.uk/competition/domains.html
b. Encode the Sokoban puzzle configuration you have chosen in PDDL2.1 level 1 (STRIPS). You may use the VAL code from the 3rd International Planning Competition (http://planning.cis.strath.ac.uk/VAL/VAL-3.1.tar.gz) to check to see if your plans are correct and the parser code to parse the PDDL domain and problem. The following resources may be helpful in understanding how to formulate PDDL domains:
    a. "An Introduction to PDDL", Malte Helmert http://www.cs.toronto.edu/~sheila/2542/f10/A1/introtopddl2.pdf
    b. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains", Fox, M. and Long, D. (2003). Journal of AI Research 20. http://www.jair.org/media/1129/live-1129-2132-jair.pdf The most relevant part of this paper is the appendix, which provides a BNF Specification of PDDL2.1.
c. Employ an existing planner to generate solutions for your test problem over a range of sizes to show the impact on performance as the problem increases in complexity. The FF planner (http://www.loria.fr/~hoffmanj/ff/FF-v2.3.tgz) is recommended for the solution generation process. The planning algorithm FF grounds all action templates by instantiating for all possible combinations of

matching values, and therefore, can be memory intensive. Our recommendation is to keep your problem size small.

d. List the following elements of your solution:
   - Operators, predicates, objects
   - The test problems (initial state, goal state)
   - A plan generated by the planner
   - Analyze the plan and comment on its quality
   - Time to arrive at a plan for three different domain sizes

Problem 3: HTN Planning

Consider the following "painting problem." We have a can of red paint (cl), a can of blue paint (c2), two paint brushes (rl, r2), and four unpainted blocks (bl, b2, b3, b4). We want to make bl and b2 red and make b3 and b4 blue. Here is a classical formulation of the problem:

S_o = {can(cl), can(c2), color(cl,red), color(c2,blue), brush(rl), brush(r2),
dry(rl), dry(r2), block(bl), block(b2), dry(bl), dry(b2), block(b3), block(b4), dry(b3), dry(b4), need-color(b1,red), need-color (b2-red), need-color(b3,blue), need-color(b4,blue)}

g = **{**color(bl,red), color(b2,red), color(b3,blue), color(b4,blue)}

Operators:
dip-brush(r, c, k)
precond: brush(r), can(c), color(c, k)
effects: ¬dry(r), canpaint(r, k)

paint(b, r, k)
precond: block(b), brush(r), canpaint(r, k)
effects: ¬dry(b), color(b, k), ¬canpaint(r, k)

Suppose, we have the following Simple Task Network: w = <paint1(t1), paintl(t2), paintl(t3), paintl(t4)>.

There is only one method:

method 1 (b, r, c, k)
task: paintl(b)
precond: need-color(b, k)
subtasks: <dip-brush(r, c, k), paint(b, r, k)>

(a) How many different possible solutions are there?
(b) How many method and operator applications will a depth-first implementation of TFD do in the best case? In the worst case?
(c) How can the domain description be modified to make the worst case more efficient?