Homework 4 – Classical Planning

TA Zae Myung Kim (zaemyung@kaist.ac.kr)
TA Jonghwan Hyeon (hyeon0145@kaist.ac.kr)

Date assigned: Wed. 28 October 2015
Date due: Fri. 6 November 2015

Submit one zip file containing your report and program codes via KLMS (naming format of the zip file: <student#>_<name>.zip, for example "20150724_john.zip"). Late submissions will not be accepted. (Write your name and student ID in your report.) This homework should be done individually and written in English.

1. The monkey-and-bananas problem (30pts)

The monkey-and-bananas problem is faced by a monkey in a laboratory with some bananas hanging out of reach from the ceiling. A box is available that will enable the monkey to reach the bananas if he climbs on it. Initially, the monkey is at A, the bananas at B, and the box at C. The monkey and box have height Low, but if the monkey climbs onto the box he will have height High, the same as the bananas. The actions available to the monkey include Go from one place to another, Push an object from one place to another, ClimbUp onto or ClimbDown from an object, and Grasp or Ungrasp an object. The result of a Grasp is that the monkey holds the object if the monkey and object are in the same place at the same height.

a. Write down the initial state description.

b. Write the six action schemas.

c. Suppose the monkey wants to fool the scientists, who are off to tea, by grabbing the bananas, but leaving the box in its original place. Write this as a general goal (i.e., not assuming that the box is necessarily at C) in the language of situation calculus. Can this goal be solved by a classical planning system?

d. Your schema for pushing is probably incorrect, because if the object is too heavy, its position will remain the same when the Push schema is applied. Fix your action schema to account for heavy objects.

2. Examine the definition of bidirectional search in Chapter 3. (20pts)

a. Would bidirectional state-space search be a good idea for planning?

b. What about bidirectional search in the space of partial-order plans?

c. Devise a version of partial-order planning in which an action can be added to a plan if its preconditions can be achieved by the effects of actions already in the plan. Explain how to deal with conflicts and ordering constraints. Is the algorithm essentially identical to forward state-space search?

Questions 3 and 4 below ask you to learn and exercise Prolog programming to solve a planning problem. Firstly, you need to download and install SWI-Prolog from http://www.swi-prolog.org, where you can also find the reference manual and some tutorials. Secondly, you must visit the website called "Prolog Tutorial" (http://www.cpp.edu/~jrfisher/www/prolog_tutorial/contents.html), which provides various sample code in Prolog.

3. Blocks World Planning in Prolog (10pts)

a. Study Section 2.19 (Actions and plans) of "Prolog Tutorial", and execute all the Prolog code in this section. Submit the entire execution log of your practice.

b. Create two or more blocks world problems of your own (i.e., by simply specifying initial and goal states different from the ones given in Section 2.19), and execute to find the solution. Submit the source code and the screen shot(s) showing the plan found.

4. Missionaries/Cannibals problem in Prolog (40pts)

(source: http://www.cs.ecu.edu/~karl/3675/fall14/assignments/assn7.html)

There are three cannibals and three missionaries on one bank of a river. They want to get to the other side. There is a rowboat, but it can only hold one or two people at a time.

The boat will not go across the river by itself. It must be rowed. Either a missionary a cannibal can row the boat, but if it ever happens that cannibals outnumber missionaries on either bank of the river, then the cannibals will eat the missionaries on that bank. (Note that it is okay for there to be some positive number of cannibals and no missionaries on a bank, since then there are no missionaries to be eaten.) Someone who is in the boat on a given bank is considered to be on that bank, so a missionary cannot hide in the boat.

The problem is to develop a plan for getting everybody across the river, without anybody being eaten.

Write a program (cannibal.pl) in a logic programming style in Prolog for the Missionaries/Cannibals problem. It should print solutions that do not involve reaching the same state twice. Print the solution as a sequence of states. It is not necessary to give instructions for how to get from one state to the next, since that should be obvious from the states themselves.

Hints:

a. Decide on a reasonable notion of a state. What do you need to know about a snapshot? How can you represent it? *This step is critical.* How you choose to represent the state can have a strong impact on how easy the program is to write. Think ahead about the operations that you need to implement, and then ask how the state might be represented to make the operations easy to write. To represent a number, you will probably find jailbird notation the best. Instead of using the number 3 to stand for 3 missionaries, for example, use list [m,m,m]. Then you can use list operations that work well in a logic programming style, instead of arithmetic operations that are problematic for logic programming.

b. Write a predicate follows(*S, T*) that is true if state *T* can immediately follow state *S* in the sequence of states. The change from *S* to *T* might involve a missionary and a cannibal paddling across the river, for example. You will find the definition of follows to be rather long. A simple approach is to use 10 axioms, one for each kind of move that can be made. (There are five kinds of moves from left to right, and five from right to left.)

   Do not worry about whether state *T* is a good state. That will be handled by a different predicate.

Note that *follows* is a predicate, and I have described it as if it is a pure test. But it will be *used* in mode follows(in, out). That is, you will give follows a state and ask it to produce a next state.

c. Write a predicate admissible(*S*) that is true if *S* is an admissible state. An admissible state is one where no missionary is being eaten on either bank of the river. Be sure to allow the case where there are no missionaries, even though there are more cannibals than missionaries.

d. A *plan* is a list of states. You will find it convenient to represent plans backwards, so that the first state in the plan is at the end of the list and the last state in the plan is at the beginning of the list. By a *plan list* I mean a list that is the reversal of a plan.

   Say that plan list *Y* is an extension of plan list *X* if there is a list *Z* such that $Y = Z ++ X$ and no state in list *Z* also occurs in list *X*. Think of *X* as a start of a plan to solve the problem, and *Z* as a continuation of it. (Remember that plans are backwards.)

   Write a predicate plan(*X, G, L*) that takes a list of states *X*, a goal state *G* and a list of states *L*, and is true if *L* is an extension of *X* that begins with state *G*. That is, in terms of the lists, $L = Z ++ X$ and $L = [G \mid Y]$. for some lists *Y* and *Z*. Predicate plan is intended to be used in mode plan(in, in, out). That is, you give it the desired partial plan and the desired goal, and it extends that partial, adding as many states as necessary to reach the goal. Since the plan list is backwards, the goal state is at the beginning.