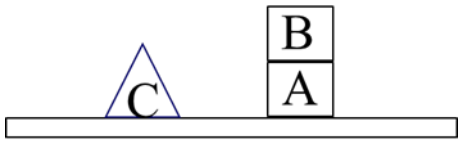
16-782, Fall'17, Planning and Decision Making for Robotics

Homework III:
Symbolic Planning for Blocks & Triangles World
DUE: Nov 15th (Wed) at 11:59PM

Description:
In this project, you are supposed to implement a planner for the blocks/triangles world. An example of this problem is shown below.



This is the same as the Blocksworld problem explained in the class except that we also have triangles that can be moved in the exact same way as blocks with the exception that nothing can be put on top of them. The number of blocks and triangles can vary and will be given as part of the input into the planner (see below).

As an input, the planner receives a set of statements that describe the start state and a set of statements that must hold true at the goal state. There are two available actions: MoveToTable(x,y) – moves x from y to table; MoveTo(x,y,z) – moves x from y onto z (note that z must be another block, not a table and not a triangle; y could potentially be a table).

As before, the planner should reside in planner.c file inside the planner() function:

```
static void planner(int* blocksV, int numofblocks, int* trianglesV,
                  int numoftriangles, int TableIndex,
                  int** onV_start, int onV_start_length,
                  int* clearV_start, int numofclear_start,
                  int** onV_goal, int onV_goal_length,
                  int* clearV_goal, int numofclear_goal,
                  int moveActionIndex, int moveToTableActionIndex,
                  int*** plan, int* planlength)
{

}
```

Currently, the body of the planner function contains a dummy planner that needs to be replaced with your planner.

The comments above the planner function in the code explain all the input parameters into the planner and its output arguments plan and planlength. As you see from the input parameters, the start state is defined by four types of statements: Block(x): x is block, Triangle(x): x is triangle, On(x,y): x is on top of y, Clear(x): there is nothing on top of x. These are passed in using arguments: blocksV, trianglesV, OnV_start, clearV_start. For example, if x is in blocksV, then it implies that x is a block (where x is an integer in the range from 0 to (numofblocks+numoftriangles) and indicates an index of an item). Similarly, if for some k, onV_start[k][0] = x and onV_start[k][1] = y, then x is on top of y at the start state. The desired configuration of the goal state is given by onV_goal and clearV_goal that are defined in the same way as onV_start and clearV_start.

The input parameters also include indices of MoveTo(x,y,z) and MoveToTable(x,y) actions. The plan returned by the planner needs to use these actions. The plan is a two dimensional array in which plan[i] corresponds to the ith action in the plan: plan[i][0] is index of that action, plan[i][1] = x, plan[i][2] = y, and plan[i][3] = z in case of MoveTo action and plan[i][3] = tableindex otherwise.

Note that the preconditions and effects for these actions are NOT defined anywhere. You need to define them yourself. You are free to introduce additional statements if you would like (even though it should be doable to write the planner using only the statements Block, Triangle, On and Clear).

To compile the C code:

```
>> mex planner.c
```

Here is an example of how to run the planner ("test1.m")

```
>> blocksV = [0 1 3];
>> trianglesV = [2];
>> TableIndex = 4;
>> onV_start = [0 1; 2 3];
>> clearV_start = [0 2];
>> onV_goal = [2 1];
>> clearV_goal = [2];
>> moveActionIndex = 0;
>> moveToTableActionIndex = 1;
>> runtest(blocksV, trianglesV, TableIndex, onV_start, clearV_start, onV_goal, clearV_goal, moveActionIndex, moveToTableActionIndex);
```

Another slightly harder example is provided in the script "test2.m".

Once your planner returns the plan, it will be printed out. While there is a very limited check of the validity of the returned plan, IT IS YOUR RESPONSIBILITY TO CHECK THAT THE PLAN IS VALID WITH RESPECT TO THE START STATE, ACTION PRECONDITIONS, ACTION EFFECTS AND GOAL STATE STATEMENTS. When grading, I will manually check that the plan returned by your program is valid.

Also NOTE: to grade your homework and to evaluate the performance of your planner, I may use a different input from the ones that are provided.

To submit:
I will receive your submissions through Gradescope and it should include

1. planner.c (or planner.cpp and other .h files you created that I need to compile)
2. ASCII file (.txt) with 1-2 paragraphs describing the approach you took for the planner (i.e., what algorithm and with what parameters).

Grading:
The grade will depend on three things:

1. How well-founded your approach is? In other words, can it guarantee completeness?
2. The quality of the plan. Is your plan optimal (minimizes the number of steps)?
3. Can your planner solve problems within SECONDS (up to 30 seconds)?