16-350: Planning Techniques for Robotics
Homework 3: Symbolic Planning
Due: April 4 (Fri), 11:59pm

Professor: Maxim Likhachev *Spring 2025 TA: Saudamini Ghatge*

1 Task

Implement a generic symbolic planner. We have provided code for reading an **environment description file** and generating the corresponding environment object. You are required to write a planner that takes an **environment object** as an input, and outputs a **sequence of actions** to go from the start to the goal. The planner function in `planner.cpp` is as follows:

```
list<GroundedAction> planner(Env *env)
{ // Your planner code }
```

2 Environment Description

An example environment description file for the Blocks world is given to you in `example.txt`:

```
Symbols:  A,B,C,Table
Initial conditions:  On(A,B), On(B,Table), On(C,Table), Block(A), Block(B), Block(C),
Clear(A), Clear(C)
Goal conditions:  On(B,C), On(C,A), On(A,Table)

Actions:
     MoveToTable(b,x)
     Preconditions:  On(b,x), Clear(b), Block(b), Block(x)
     Effects:  On(b,Table), Clear(x), !On(b,x)

     Move(b,x,y)
     Preconditions:  On(b,x), Clear(b), Clear(y), Block(b), Block(y)
     Effects:  On(b,y), Clear(x), !On(b,x), !Clear(y)
```

`Env` class:

In the provided code, we generate an environment object (of the `Env` class) from the environment description file. The `Env` class includes the (1) initial conditions, (2) goal conditions, (3) actions, (4) symbols. An object of the `Env` class is passed to your planner. Your environment description file should follow the same template as `example.txt`.

The `Env` class uses the data structures below. You may add more functions to them as needed. **However, DO NOT change the `main` function!**

- `Condition`: this class includes 3 variables: (1) name of the condition, (2) the arguments, (3) if the condition is negated or not.
- `GroundedCondition`: this class includes 3 variables: (1) name of the condition, (2) the values for the arguments, (3) if the condition is negated or not.
- `Action`: this class includes 4 variables: (1) name of the action, (2) action arguments, (3) preconditions, (4) effects.
- `GroundedAction`: this class includes 2 variables: (1) name of the action, (2) values for the arguments.

You are required to write a generic planner that outputs a sequence of steps to go from the initial condition to the goal condition. The output of your planner is a list of GroundedActions.

Environments:

1. **Blocks and Triangles World:** This environment is similar to the Blocks world problem explained in the class. In addition to the blocks, this environment has triangles that can be moved in the exact same way as blocks with the exception that nothing can be put on top of them. A simple example of this environment with only three objects is shown below.

   We provide a description file for an environment with 5 blocks (`B0, B1, B2, B3, B4`), 2 triangles (`T0, T1`) and a `Table`. The start and goal conditions are:

   - Start conditions: `B0` is on `B1, B1` is on `B4, B2` is on `Table, B3` is on `B2, B4` is on `Table, T0` is on `B0,` and `T1` is on `B3`.
   - Goal conditions: `B0` is on `B1, B1` is on `B3,` and `T1` is on `B0`.

2. **Fire Extinguisher Environment:** In this environment, a pair of robots must put out a fire. This domain has two robots: (1) a quadcopter, and (2) a mobile robot. The mobile robot can travel between locations. The quadcopter cannot travel between locations by itself, and cannot land on the ground. The quadcopter can only travel between locations by landing on the mobile robot and having the mobile robot travel to the other location. The quadcopter can hover around a single location if its battery level is High, but it won't be able to take off if its battery level is Low. Whenever the quadcopter is on the mobile robot, it can charge its battery by calling the charge action. The quadcopter has a tank. This tank can be filled with water when the quadcopter is on the mobile robot at location `W` (where there is water). The fire is at location `F`. The `W` and `F` locations are far from each other. The quadcopter must hover around location `F` in order to pour water on the fire. The quadcopter needs to pour water on the fire three times in order to extinguish the fire. Every time the quadcopter pours water on the fire, its battery level becomes low and its water tank becomes empty. It would have go back to `W` to fill its tank again. The robots will each start at one of five different locations (`A, B, C, D, E`), which are far from `W` and `F`. The start and goal conditions are:
    - Start conditions: the quadcopter is hovering around location `B`. The mobile robot is at location `A`. The quadcopter's water tank is empty.
    - Goal: The fire is extinguished.

3 Execution

To compile the cpp code (add `-std=c++11` if needed):

```
>> g++ planner.cpp -o planner.out
```

To execute:

```
>> ./planner.out example.txt
```

Replace `example.txt` with the description file of the environment you want to plan for.

Once your planner returns the plan, it will be printed out. It is your responsibility to check whether the plan is valid with respect to the start conditions, actions, and goal conditions.

NOTE: you should write a domain-independent planner. The environment object is passed into your GENERIC (domain-independent) planner which just runs a search by applying available valid actions to every state. Your code will be tested with other environments. In summary, you should 1) write a domain-independent planner that generates a plan for any environment that follows the strips representation, and 2) include the discussion and results (i.e., planning time and number of expanded states) of your generic planner applied on the three environments: a) Blocks b) Blocks and Triangles and c) Fire Extinguisher.

4 Submission

You will submit this assignment through Gradescope. You must upload one ZIP file named `<andrewID>.zip`. This should only contain:

1. A folder `code` that contains 1) a single executable named `planner.out`, 2) all the C++ source files for the planner, and 3) the description files for the three environments. Your code should handle relative paths.

2. Your writeup in `<andrewID>.pdf`. This should contain a summary of your approach for solving this homework, results, and instructions for how to compile your code. Do not leave any details out because we will **not** assume any missing information. Include the time that the planner takes and the number of states that the search expands for each of the three environments with and without a heuristic.

    - You are required to implement at least one heuristic. You must compare results obtained with and without heuristics.
    - For all cases, i.e., for all environments and all planners (with and without heuristics), report and discuss the following:
        - Number of steps in the plan.
        - Number of states expanded by your search.
        - Amount of time taken by the planner.

Please make sure it does not contain any other stray files like `.git` and MAC OS auto-generated files.

5 Grading

Your grade will depend on:

1. How well-founded your approach is. Can your planner guarantee completeness?

2. Whether your planner is domain-independent. Is it implemented as a generic search that can be used to solve a completely different problem?

3. The quality of the plan. Is your plan optimal (minimizes the number of steps)? Can your planner solve problems within 30 seconds?

4. The quality of your write-up. Provide an explanation for all reported statistics. Provide a discussion of the pros and cons of using a heuristic.

6 Extra Credits

You may earn extra credit by completing the following:

Design an entirely new environment (not one of the three domains mentioned above), add the corresponding configuration file, describe it, and present the performance of the planner on it. We will run your planner on it to verify. (10 points)