

Problem Description

In this homework, you will implement a generic symbolic planner. We have provided the code for reading an environment description from a file by using regular expressions and generating the corresponding environment object. Your job is to write a planner that gets an environment object as an input and outputs a sequence of actions to go from the start to the goal. An example of the environment description for the Blocks World that was taught in the class is below:



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)

In the provided code, we parse the description files for you and provide you with the environment object (Env class) which includes the 1) initial conditions, 2) goal conditions, 3)

actions, and 4) symbols. An object of the Env class is passed to your planner.

The Env class uses the data structures below. Feel free to add more functions to them as needed. However, **do not** change the main function.

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

In this homework, we provide the environment description files for three environments: 1) Blocks World, 2) Blocks and Triangles, and 3) Fire Extinguisher.

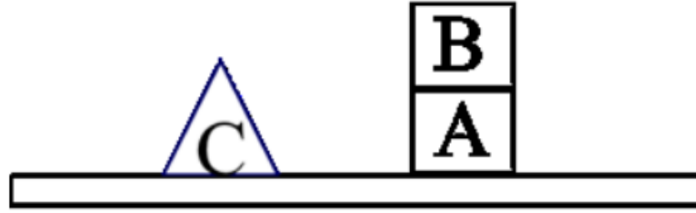
We will explain these environments later on. These environment description files are parsed and an environment object (Env) is passed to your planner. Your job is to write a general planner that outputs a sequence of steps to get from the initial condition to the goal condition. The output of your planner is a list of GroundedActions (`std::list<GroundedAction>`).

Environments

The Blocks World environment was introduced earlier. Details for the remaining two are below.

Blocks and Triangles Environment

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 below:

- 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.

For easier debugging, we provide a trivial environment (BlocksEasy.txt) with 3 blocks (A,B,C).

- Start conditions: A is on B, B is on Table, C is on Table.
- Goal conditions: A is on Table.

Fire Extinguisher Environment¹

The goal of this problem is to have a pair of robots 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 only moves between locations by landing on the mobile robot and having the mobile robot travel to the other location. The quadcopter can fly around a single location (cannot navigate between locations) 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 that 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 should fly 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.

¹Inspired by the final challenge at 1st Summer School on Cognitive Robotics at MIT.

Every time the quadcopter pours water on the fire, its battery level becomes low and its water tank becomes empty (it should go back to W to fill its tank). The robots will each start at one of five different locations (A, B, C, D, E), which are far from W and F. The quadcopter cannot land on the ground.

- Start conditions: the quadcopter is flying and at location B. The mobile robot is at location A. The quadcopter's water tank is empty.
- Goal: The fire is extinguished.

Compiling and Executing the Code

Open a terminal and navigate to the `code` directory. Then:

```
>> mkdir build
```

```
>> cd build
```

Building with Cmake

```
>> cmake ..
```

```
>> cmake --build . --config Release
```

To build in debug mode change `Release` with `Debug`. Now, you can run the code with:

```
>> ./planner [envName]
```

Example:

```
>> ./planner Blocks.txt
```

You can find the environments in the `envs` directory.

Building with g++

To compile on Linux (Mac or Windows may require a substitute for g++, e.g. clang):

Make sure you are within the `build` directory as shown above. Then: `>> g++ ../src/planner.cpp -o planner` (optional but may be necessary: `-std=c++11`)

To enable debugging, add a `-g` tag:

```
>> g++ ../src/planner.cpp -o planner -g
```

This creates an executable, namely `planner`, which we can then call with different inputs:

```
>> ./planner [envName]
```

Example:

```
>> ./planner Blocks.txt
```

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.

Note: You should write a domain-independent planner. The environment object is passed into your **generic** (domain-independent) planner which runs a search by applying available valid actions to every state. **Your code will be tested with other environments.**

Submitting Your Work

Please submit your work through Gradescope. Your submission should include:

- A folder named `code` that contains 1) all the C++ source files for the planner and 2) the description files for the three environments.
- A PDF writeup named `<Andrew ID>.pdf` with instructions to compile code, results, and everything we need to know about your implementations and submission. 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 (no need to include `BlocksEasy`) with and without a heuristic.

Grading

Your grade will depend on:

- How well-founded your approach is. In other words, can your planner guarantee completeness?
- How domain-independent the planner is. That is, is it implemented as a generic search that can be used to solve a completely different problem from a different domain?
- The quality of the plan. Is your plan optimal (minimizes the number of steps)?
- Planning speed. Can your planner solve problems quickly (less than 30 seconds)?
- The quality of your writeup. Provide a discussion on how much improvement you get when you use a heuristic in your search by comparing it with a planner that does not use any heuristics.

Extra Credit

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)
- Implement the empty-delete-list heuristic as described in the lectures, evaluate its performance as a function of the size of the problem (e.g., number of blocks and triangles), and describe the results of your evaluation. Explain how to enable this heuristic, so that we can run it during the grading. (10 points)