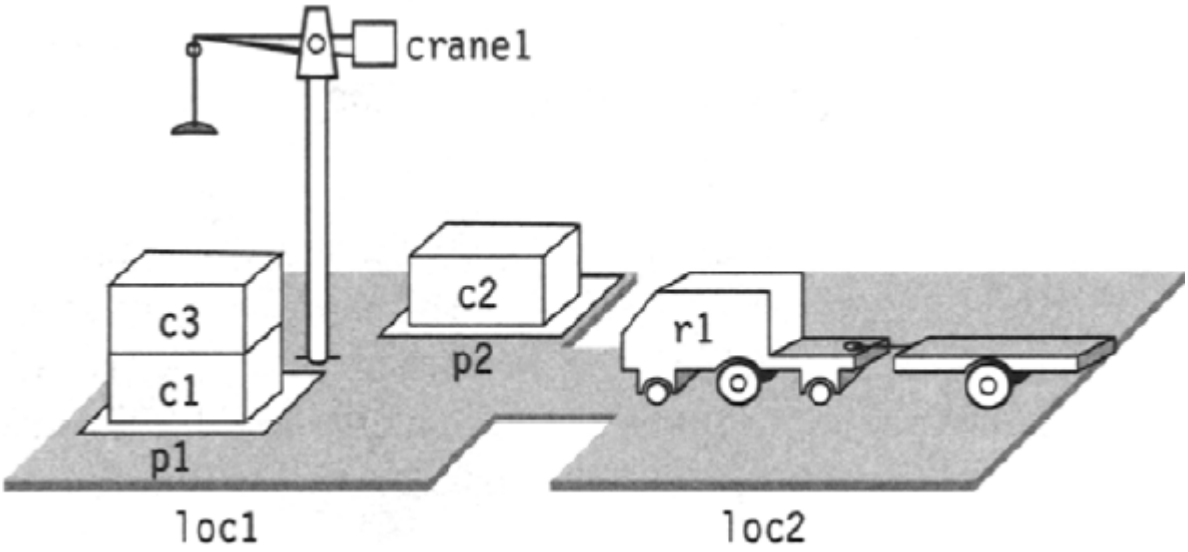
Assignment 2: Planning
EECS 492: Artificial Intelligence
Fall 2015

Now: Thursday, October 1, 10:30 am
Due: Thursday, October 22, 11:00 pm



You are a summer intern at ACME shipping company, which owns a harbor with cargo ships, cranes, roads, robots, parking lots, and storage piles for cargo containers. They ask you to write software that automatically generates plans, which will be executed by the robots, for moving the containers from the cargo ships to their designated storage locations. This situation can be modeled formally in Planning Domain Definition Language (PDDL) as:

- A set of *locations* {`l0, l1, l2 ...` } that correspond to docked ships, storage areas, and parking areas.
- A set of *robots* {`r0, r1, ...` }
- A set of *cranes* {`k0, k1, ...` }
- A set of *piles* {`p0, p1 ...`}
- A set of *containers* {`c0, c1, ...` }
- A symbol `G` that denotes the ground, on which all piles rest.

The topology of the harbor, which cannot be modified, is specified using three predicates:

- `adjacent(l, l')` : location l' is adjacent to location l
  - Note: this only specifies that there exists a one-way road from l to l'. To specify a two-way road, we must also add `adjacent(l', l)` to the list of initial conditions.
- `attached(p, l)` : Pile p is attached to location l

- `belong(k, l)` : Crane k belongs to location l

The arrangement of trucks and shipping containers, which can change over time, is specified by the following:

- `occupied(l)` : Location l is occupied by a robot
- `free(l)` : Location l is not occupied by any robot
- `at(r, l)` : Robot r is at location l
- `loaded(r, c)` : Robot r is loaded with container c
- `unloaded(r)` : Robot r is not loaded with any `container`
- `holding(k,c)` : Crane k is holding container c
- `empty(k)` : Crane k is not holding a container
- `in(c, p)` : Container c is in pile p
- `on(c, c')` : Container c is on top of c', which is either a container or `G`
- `top(c, p)` : Container c sits on top of pile p. If pile p is empty, we write `top(G, p)`

Note that these predicates are not independent. For example, `unloaded(t)` can only be true if there is no c such that `loaded(t,c)`.

Next we specify the five possible actions: `move, load, unload, put, take`.

```
move(r, l, m) # move robot r from location l to location m
        precond: adjacent(l, m), at(r, l), free(m)
        add: at(r, m), occupied(m), free(l)
        delete: occupied(l), at(r,l), free(m)
load(k, l, c, r) # crane k at location l loads container c onto robot r
        precond: belong(k,l), holding(k,c), at(r,l), unloaded(r)
        add: empty(k), loaded(r,c)
        delete: holding(k,c), unloaded(r)
unload(k,l,c,r) # crane k at location l takes container c from robot r
        precond: belong(k,l), at(r,l), loaded(r,c), empty(k)
        add: holding(k,c), unloaded(r)
        delete: empty(k), loaded(r,c)
put(k,l,c,d,p) # crane k at location l puts c onto d in pile p
        precond: belong(k,l), attached(p,l), holding(k,c), top(d,p)
        add: empty(k), in(c,p), top(c,p), on(c,d)
        delete: holding(k,c), top(d,p)
take(k,l,c,d,p) #crane k at location l takes c off of d in pile p
        precond: belong(k,l), attached(p,l), empty(k), top(c,p), on(c,d)
        add: holding(k,c), top(d,p)
        delete: empty(k), in(c,p), top(c,p), on(c,d)
```

Assignment

Your job is to fully implement a Systematic Nonlinear Planner for this domain. You may write in either C++ or python. You may use no libraries besides the C++ standard library, or standard python I/O modules. The planner should take two arguments: an input file specifying a problem instance, and an output file.

Resources for the SNLP Planning Algorithm

Planning, SNLP planning, and the unification algorithm are described in the following texts:

- Section 10.1 and 10.2 in the course textbook (AIMA 3e), which describe classical planning.
- Section 9.1 and 9.2 in the course textbook, which describe lifting and the unification algorithm you will need to implement.
- The classic 1991 paper *"Systematic Nonlinear Planning"* by McAllester and Rosenblitt, which describes the core SNLP algorithm. This is provided on CTools.
- The paper *"Systematic Nonlinear Planning: A Commentary,"* in which Dan Weld, a leading expert on planning systems, nominates the 1991 paper for the AAAI-10 Classic Paper Award. Provided on CTools.
- Section 11.3 in the previous edition of the textbook (AIMA 2e), which provides a textbook introduction to SNLP Planning. Provided on CTools. Be sure to look at problem 11.11.

Input file format

The input file (see the example at the end of the spec) will contain the following in order, separated by newlines:

- The string "locations" followed by an integer specifying the number of locations
- The string "robots" followed by an integer specifying the number of robots
- "cranes" followed by the number of cranes
- "piles" followed by the number of piles
- "containers" followed by the number of containers
- "initial" followed by a list of initial conditions.
  - Conditions are identified by the following set of keywords: {`adjacent, attached, belong, occupied, free, at, loaded, unloaded, holding, empty, in, on, top`}
  - Each keyword is followed by a list of arguments, separated by whitespace. The arguments each consist of either a character followed by a number, or the symbol `G`. The characters are {`l, r, k, p, c`} for location, robot, crane, pile, and container respectively. The number specifies which instance of each object is being dealt with. **Objects are zero-indexed for your convenience.**
  - For example: if there are 4 locations and 2 robots, then we have access to the symbols {`l0, l1, l2, l3, r0, r1`}.

- "goal" followed by a list of goal conditions, separated by newlines.
- Comments are preceded by the symbol "#" and end at the end of the line. The first five lines of the file must be exactly as stated above, with no comments.

Output file format

The output file (see the example at the end of the spec) is formatted similarly. It consists of:

- The string "actions" followed by a list of actions separated by newlines, in no particular order.
  - Actions are one of the following: {`start, finish, move, take, put, load, unload`}.
  - Each action is labeled by a unique nonnegative integer.
  - Thus we could write `..., 23 move r2 l3 l4, 17 take c1 k0 p3, ...` to indicate that action 23 moves robot `r2` from location `l3` to location `l4`, and action `17` uses crane `k0` to take container `c1` from pile `p3`. The action numbers do not indicate any ordering; their only purpose is to uniquely identify actions.
- The string "constraints:" followed by a list of constraints separated by newlines.
  - Constraints consist of pairs of action numbers separated by the symbol "`<`"
  - For example, "`23 < 17`" indicates that action `23` is to be performed before action `17`.
  - Since every action must be bound by the constraints that it comes after `start` and before `finish`, you do not need to explicitly print these ordering constraints.

Compiling and Running

For C++, we will use the following commands with g++ 4.6 to compile and run your code:

```
g++ -std=c++11 -o planner <source1.cpp> <source2.cpp> ... <sourceN.cpp>
./planner <inputfile> <outputfile>
```

For Python, we will run the following command with Python 2.7:

```
python planner.py <inputfile> <outputfile>
```

"Inputfile" is a file containing an instance of the planning problem, and "outputfile" should contain a valid plan.

Provided code, test cases, and executables

We will provide you with the following:

- 5 of the 10 test cases your code will be run against
- An executable that checks whether or not an output plan is a solution to the given input file
- Data structures for variables, predicates, and actions
- Functions that read input files and print plans

What to turn in

You must submit a zip file containing:

1. All your source and header files in a folder called "code".
   (a) If you use python, your main function should be in a file called "planner.py"
2. A pdf with answers to the following questions:
   (a) What major functions did you write, what do they do, and why? You can ignore less relevant helper functions. (Write a brief paragraph per function.)
   (b) What search algorithm did you use, and why? If you used heuristics, what were they? (Write at least a paragraph).
   (c) Devise your own real-world planning problem and give a full PDDL description. Which planning or search algorithm would you use to solve it?

Grading

Grading is out of 100 points.

- Test cases: 60 pts
  - We will run your code on 10 test cases (5 of which will be provided to you in advance), 6 pts per test case
  - 3 pts for generating a working plan
  - 3 pts for no obvious inefficiencies, such as
    * Solution far longer than necessary
    * Moves that immediately undo each other (i.e., moving a robot back and forth without doing anything else)
    * Taking an excessive amount of time to reach a solution
- PDF and code: 40 pts
  - 5 pts for code compiling and running without error
  - 10 pts for code description
    * Criteria: all major functions documented, with clear and technically correct explanations.
  - 10 pts for description of reasonable search algorithm
    * Criteria: clear and technically correct description of a search algorithm, and any heuristics used.
  - 15 pts for your own planning problem, PDDL description, and algorithm choice
    * Criteria: problem is not too trivial, and a reasonable PDDL description is given, with a sensible algorithm choice.

Hints and Tips

- A partial plan is a solution if all of the following hold:
  - There are no open preconditions
  - Actions are order consistent
  - There are no variable inequalities of the form $x \neq x$
  - There are no threats
- In the McAllester & Rosenblitt paper, the term *nondeterministically* means that more than one successor plan can be generated at that step, and you should search all of them.
- Making a copy of an expression with fresh variables is the same as "standardizing apart."
- Variable equalities can be tracked by applying a substitution everywhere in the existing plan. That is, if we determine $x = y$, then we substitute all instances of y for x in the plan. Otherwise, tracking equalities can be very messy.
- Your program does not need to handle malformed inputs, disjunctions of preconditions or effects, or negated preconditions.

Input Example

```
locations 4
robots 1
cranes 2
piles 3
containers 2
# l0 is the dock / docked ship, l1 is the robot parking lot,
# l2 is an intermediate location, l3 is the storage area
# l0 has one pile, l3 has two. Each site has a crane,
# which starts out empty.
# The robot starts out at l1.

initial
adjacent l1 l0
adjacent l0 l1
adjacent l0 l2
adjacent l2 l0
adjacent l2 l3
adjacent l3 l2
attached p0 l1
attached p1 l3
attached p2 l3
belong k0 l1
belong k1 l3
occupied l1
at r0 l1
```

```
unloaded r0
empty k0
empty k1
in c0 p0
in c1 p0
on c0 G
on c1 c0
top c1 p0
#don't forget to add the following for empty piles
top G p1
top G p2

# The goal is to have all containers in location l3,
# one in each pile, with the robot parked back at l1.
goal
in c0 p1
in c1 p2
at r0 l1
```

Output Example

```
# A solution to the problem defined in the input file.
actions
0 start
1 finish
2 move r0 l1 l0
3 take k0 l0 c1 c0 p0
4 load k0 l0 c1 r0
5 move r0 l0 l2
6 move r0 l2 l3
7 unload k1 l3 c1 r0
8 put k1 l3 c1 G p2
9 move r0 l3 l2
10 move r0 l2 l1
11 take k0 l0 c0 G p0
12 load k0 l0 c0 r0
13 move r0 l0 l2
14 move r0 l2 l3
15 unload k1 l3 c0 r0
16 put k1 l3 c0 G p1
17 move r0 l3 l2
18 move r0 l2 l1
19 move r0 l1 l0

# Note 1: All the constraints involving start / finish,
# such as 0 < 1, 0 < 2, 0 < 3, 2 < 1, 3 < 1, etc.,
# are implied and do not need to be printed
# Note 2: in this example the actions happen to be linearly ordered.
```

```
# This will not be true in general.
constraints
2 < 3
3 < 4
4 < 5
5 < 6
6 < 7
7 < 8
8 < 9
9 < 10
10 < 11
11 < 12
12 < 13
13 < 14
14 < 15
15 < 16
16 < 17
17 < 18
18 < 19
```