AIP – part 5: building operators

brief

You are required to build a small collection of state changing operators in the style used in lectures. Each operator will have preconditions, additions, deletions and a textual descriptor. You may invent new types of tuple as required by the problem you are given.

All work is group work, using the same groups as in previous earlier work (unless advised otherwise) and will use the same approach for peer assessment. Each group will be assigned one of the problem areas below and given a time limit (2-3 weeks) to complete the work.

Problems are outlined below (see tutors for clarification if necessary). Your final collection of operators should work with the mops-search mechanism. You will be expected to demonstrate your operators using sample start & goal states. You should show the operation of mops-search with at least 3 start/goal combinations.

When designing your operators think about "what should be possible" not just what you will want them to do during a given test. Advanced solutions will also consider efficiency aspects of their operators.

The problems specify some operators which you should build, you may also build others if this helps your solution.

an example operator

See below for an example operator, check lecture materials for more examples .

```
(def move-op
  '{:txt [?agent moves from ?place1 to ?place2]
    :pre ([isa ?agent agent]
          [at ?agent ?place1]
          [next-to ?place1 ?place2])

    :del ([at ?agent ?place1])
    :add ([at ?agent ?place2])
    })
```

problem areas

sliding doors

This scenario is concerned with doors. Doors can be in different states (open or closed, locked or unlocked). Doors connect rooms and are un/locked with specific keys.

specify & build the following operators...

- open (an unlocked door)
- close (a door)
- lock (a door with a key)
- unlock (with a key)
- move (from one room to another)

lifting

This scenario is based around using a lift. Agents can "call" a lift by pressing the button outside the lift door in a corridor, on calling a lift there will be a short delay then the lift will arrive and the doors will open. Once entering the lift a floor can be selected (using a button inside the lift). After selecting a floor, the lift doors will close & it will start to move. After a short delay it will reach the chosen floor & the doors will open. You may assume that your operators control all agents using the lift.

specify & build the following operators...

- call-lift (from the corridor, assume that there is only one button per floor, not separate buttons to request a lift going up and another going down)
- select-floor (by pressing a button inside the lift)
- enter-lift (from the corridor)
- exit-lift (to the corridor)
- wait (for a lift after it is called)
- wait (for a lift to reach the selected floor)

climbing

Some objects can be held by agents (grabable objects), some can be platforms and some can be climbed, some platforms can be climbed and some climbable and/or platform objects are also grabable.

A platform object can have other objects placed on top of it.

Agents can pick/drop grabable objects off/on an platform if they are standing on the platform.

specify & build the following operators...

- climb-on (a climable object at roughly the same position as the agent)
- climb-off (a climbable object)
- pick-off (an object from a platform)
- drop-on (an object on a platform)

NB: you must also be able to test your work using pick-up, drop & move (either as they were defined in lecture materials or with some modification).

cranial loading

There is a small container terminal in Gliwice, Poland where containers are loaded/unloaded to/from boats. In 2010/11 researchers were investigating using STRIPS operators to automate this process. That much is true, the rest of this is fiction...

Boats arrive at one side of the terminal (one at a time), trains arrive at the other side. Boats & trains both have a line of places/platforms/locations where containers can be stacked.

The line of platforms on a boat/train are described with tuples like...

```
[platforms train-1 (t1 t2 t3 t4 t5 t6)]
```

A crane is positioned between the boat arrival side and the train arrival side. The crane can rotate but cannot move forward/backward along the terminal. Boats and trains can move, when train-1 is positioned to load/unload to/from platform p3, this will be represented by the tuple...

```
[position train-1 t3]
```

Boats & trains can move forwards & backwards (your operators should handle this but they do not need to deal with boats/trains moving off the terminal – this is managed by other operators).

Containers are stacked platform positions (for the sake of this exercise you can assume that there is no limit to the height of stacks). When the crane lifts a container it removes it from the top of a stack, when it places a container, it puts it on the top of a stack. Stacks are represented by the following type of tuple...

```
[stack t3 (a b c)]
```

...where a, b & c are containers (a is on the top of the stack).

specify & build the following operators...

- lift-container – picks up a container from the top of the current stack
- place-container– puts a container onto the current stack
- rotate-crane – switch the crane from the boat side to the train side & vice-versa
- shunt-fwd – move a boat/train forward one platform/position (do nothing if this already at the last position in the sequence)
- shunt-back – move a boat/train backward one platform/position (do nothing if this already at the last position in the sequence)

robot cars

In this example robot cars move stock materials around a small warehouse.

The warehouse is split into zones with one car in each zone. Cars cannot move between zones but it is possible to move stock from one zone to another because zones connect to each other by *Exchanges* – shared areas where one car can leave stock for another (note there is only room for one piece of stock in an exchange).

Each zone is laid out in a grid formation where *Corridors* run north-south or east-west and connect to each other at *Junctions*. The corridors are full of *Bays* – the bays are where stock is stored. So far so good but there is a small complication... the stock is mostly (you can assume always) longer than the width of the corridors so they can only be carried along the corridors if they correctly aligned this means that cars sometimes need to rotate stock at corridor junctions.

specify & build operators to achieve the following...

- collect-stock - from the car's current bay/exchange (cars must already have travelled to the correct bay), note that cars can only hold one stock item at a time (you may split this into 2 operators if you wish);
- deposit-stock - put the stock at the bay/exchange where the car is (you may split this into 2 operators if you wish);
- move-to-bay - move a car to a bay in its current corridor;
- move-to-junction - move a car to a junction/exchange from its current corridor (you may combine this with other move operators if you wish)
- rotate-car - rotate a car (and its stock) this can only happen at a junction

You will also need to specify world knowledge (static/unchanging tuples) to capture details about which junctions connect to which corridors, where the bays are and the location of exchanges.

NB: ensure your definitions describe a world large enough to test but do not make it larger than necessary. If necessary you can write small helper functions to generate tuples from other data types.