Planning and Scheduling, @Home Project: Storing Groceries

Linda Koine, Pascal Maczey and Oussama El Mahboubi

18. Januar 2019

Inhaltsverzeichnis

1 Planner

1.1 Choice of Planner

We chose to use shop2 for the planning project, because it is sound and complete (GA14). We want to trust the correctness of the plans to be returned. The planner UMCP is also sound and complete, but there are some use cases, in which shop2 performs better than UMCP (GA14).

1.2 Installation

Shop2 can be downloaded as a lisp version and as a java version [1]. First we wanted to use the java version of shop2 because we haven't worked with lisp before. But the installation didn't worked either on a linux system, a windows system or a mac system. Most likely the reason why the installation of the java version didn't work was an error in the makefile but we haven't done an deep analysis of the problem. Thus we decided to use the lisp version of shop2. The installation on linux and mac wasn't complicated. On windows we first tried to get it to run with LispWorks but didn't get it to run correctly. After installing clisp and using the terminal it worked too.

To run the Shop2 2.9 lisp version quicklisp is needed. To run quicklisp we need to call

```
    sbcl --load quicklisp.lisp
```

After that we need to call

```
    (ql:quickload "shop2")
```

and

```
    (load "<path to file to be run>")
```

. There are several examples that show the features of shop2.

2 Working with SHOP2

The task is to take objects of a table and put the objects on shelves of a cupboard. The table, cupboard, shelves and objects may have to be located and the door of the cupboard may be closed in the beginning.

[1] https://sourceforge.net/projects/shop/files/

2.1 General approach

In general for shop2 to generate a plan a domain and a problem description is needed. In the domain description are the operators, methods and axioms that can be used to generate a plan. By defining the methods domain specific knowledge can be included. Methods can be decomposed into operators and axioms can be used as preconditions. The problem description contains the initial state (some logical atoms) and the start-task that should be decomposed into a plan.

2.2 Explicit planning domain description

By using operators, methods and axioms it is possible to describe how the task „put objects into cupboard“ can be done. The description of the general approach used here follows now:

The starting task is called *start*. The task gets as parameters the table and the cupboard, so that if there are more tables than one we know which one to use. The task *start* decomposes into other tasks, so that the robot first preserves the room: It finds its current location and the locations of the table and the cupboard if necessary. When it walks to the cupboard and opens its door if it is closed, so that we do not have to check if it is open every time we put an object into the cupboard. We assume there are no effects on our system, so nobody closes the cupboard-door while the plan is executed. We find all locations of the shelves in the cupboard, so that we know the status of the whole cupboard is necessary.

Now we can walk to the table and search for objects that should be moved. A object may have a category (as each shelf). Each object has information if it has to be moved. Also it is possible to say that all objects of a certain category should be moved. If there is no object on the table that should be moved the robot stops. Else it takes one of the objects and checks again if there are still objects on the table that should be moved. If there is at least one object left the robot carries the object it picked up into the cupboard and comes back to the table. If there is not an object left on the table it doesn't come back to the table and stops after putting the last object in the cupboard. This methods are called recursively until there is no object left on the table that should be moved.

The objects are placed on the shelves in the cupboard, so that the categories fit. If there is no category the object is placed anywhere.

The domain modulation does not limit the number of objects that the table and the shelves can hold. The planning fails, if there is no shelf that has the same category like an object that has to be moved. Because of the way the methods are constructed we only are able to move one object at a time.

2.3 Explicit problem description and planning results

The problem description has to contain the initial state. A very simple initial state is the following:

```
    (defproblem case0 atHome
    (
    (knowLocation table)
    (knowLocation cupboard)
    (knowLocation shelf)
    (close cupboard)
    (at table)
    (isTable table)
    (isCupboard cupboard)
    (has(cupboard shelf))
    ) ((start table cupboard)))
```

Here we do not have any objects on the table. We also know all of the locations of the table, the cupboard and its shelf. The cupboard is closed. The robot is located at the table in the beginning. The plan produced for this initial state is the following :

```
    (((!CHECKCURRENTLOCATION TABLE)
    (!WALKFROMTO TABLE CUPBOARD)
    (!OPEN CUPBOARD)
    (!FOUNDALLSHELFS CUPBOARD)
    (!WALKFROMTO CUPBOARD TABLE)
    (!CHECKNOOBJECTTOBEMOVEDLEFT TABLE)
    (!END)))
```

In the next case addes one object on the table that should be moved:

```
    (defproblem case1 atHome
    ( (on(obj table))
    (knowLocation obj)
    (knowLocation table)
    (knowLocation cupboard)
    (knowLocation shelf)
    (close cupboard)
    (at table)
    (isTable table)
    (isCupboard cupboard)
    (has(cupboard shelf))
    (objectShouldBeMoved obj)
    ) ((start table cupboard)))
```

This situation description is similuar to the first case of the @Home-Project. The plan produced for the *case1*-problem description is the following:

```
    (((!CHECKCURRENTLOCATION TABLE)
    (!WALKFROMTO TABLE CUPBOARD)
    (!OPEN CUPBOARD)
    (!FOUNDALLSHELFS CUPBOARD)
    (!WALKFROMTO CUPBOARD TABLE)
    (!TAKEOFF OBJ TABLE)
    (!CHECKNOOBJECTTOBEMOVEDLEFT TABLE)
    (!WALKFROMTO TABLE CUPBOARD)
    (!PUTON OBJ SHELF)
    (!END)))
```

After the robot opens the cupboard and carries it to the cupboard and puts it onto the only shelf.

Now we take a look at a more complicated problem description. The location of the table, the cupboard and shelves are not known. The robot is placed somewhere in the room in the beginning. There are five objects on the table with two different categories. All objects with the category one should be moved. Also *obj3* and *obj1* are marked explicitly, so that the robot knows that they should be moved. So *obj2* and *obj4* are not to be taken to the cupboard. This situation description is similar to the fourth case of the @Home-Project.

```
  (defproblem case4 atHome
  (
  (on(obj1 table))
```

```
(on(obj2 table))
(on(obj3 table))
(on(obj4 table))
(on(obj5 table))
(at somewhereInRoom)
(has(cupboard shelf1))
(has(cupboard shelf2))
(category shelf1 1)
(category shelf2 2)
(close cupboard)
(isTable table)
(isCupboard cupboard)
(category obj1 1)
(category obj2 2)
(category obj3 2)
(category obj4 2)
(category obj5 1)
(objectShouldBeMoved obj3)
(objectShouldBeMoved obj1)
(categoryShouldBeMoved 1)
) ((start  table cupboard)))
```

The plan produced for the *case4*-problem description is the following:

```
  (((!CHECKCURRENTLOCATION SOMEWHEREINROOM)
  (!FINDLOCATION TABLE)
  (!FINDLOCATION CUPBOARD)
  (!WALKFROMTO SOMEWHEREINROOM CUPBOARD)
  (!OPEN CUPBOARD)
  (!FINDLOCATION SHELF1)
  (!FINDLOCATION SHELF2)
  (!FOUNDALLSHELFS CUPBOARD)
  (!WALKFROMTO CUPBOARD TABLE)
  (!FINDLOCATION OBJ1)
  (!TAKEOFF OBJ1 TABLE)
  (!CHECKSTILLOBJECTTOBEMOVEDLEFT TABLE)
  (!WALKFROMTO TABLE CUPBOARD)
  (!CHECKIFCATEGORYFIT OBJ1 SHELF1)
  (!PUTON OBJ1 SHELF1)
```

```
    (!WALKFROMTO CUPBOARD TABLE)
    (!FINDLOCATION OBJ3)
    (!TAKEOFF OBJ3 TABLE)
    (!CHECKSTILLOBJECTTOBEMOVEDLEFT TABLE)
    (!WALKFROMTO TABLE CUPBOARD)
    (!CHECKIFCATEGORYFIT OBJ3 SHELF2)
    (!PUTON OBJ3 SHELF2)
    (!WALKFROMTO CUPBOARD TABLE)
    (!FINDLOCATION OBJ5)
    (!TAKEOFF OBJ5 TABLE)
    (!CHECKNOOBJECTTOBEMOVEDLEFT TABLE)
    (!WALKFROMTO TABLE CUPBOARD)
    (!CHECKIFCATEGORYFIT OBJ5 SHELF1)
    (!PUTON OBJ5 SHELF1)
    (!END)))
```

We see, that *obj2* and *obj4* are left on the table.

2.4 Performance - SHOP2

By running the *run.lisp* all test-cases are called multiple times, including the *case0* where nothing is on the table. The *run.lisp* and the different cases are in the *ATHOME*-directory. If we start qicklisp and shop2 in that directory we can call:

```
    (load "run.lisp")
```

By using *:verbose :plans :which :FIRST* the first plan found will be returned. The sequence of methods in the *athome.lisp*-file is made up such a way, so that at first the most general method is tried out to produce a plan. In that way most times if we try out a method we can stick with that one and do not have to try out others. This reduces the time we need to find a plan. Also, this makes the first plan found sometimes optimal. So by ordering the methods in the lisp-file we include domain specific knowledge. Shop2 gives information about the plans found that are listed in the following table:

	Plans	cost	Expansions	Inferences	CPU time	Real time
case0	1	7.0	20	41	0.000	0.001
case1	1	10.0	37	112	0.001	0.001
case2	1	32.0	78	330	0.002	0.002
case3	1	37.0	84	362	0.003	0.003
case4	1	30.0	74	409	0.002	0.002

The cost of the use of one operator is always 1.0, that means we can use the costs to find out how long the plan is in the end. The time given is in seconds. The more complex a problem description is the more expansions shop2 needs to find a plan and the more time the planning takes. Still, we find good plans very fast.

Literatur

[GA14] Georgievski, Ilce ; Aiello, Marco: An Overview of Hierarchical Task Network Planning. In: *CoRR* abs/1403.7426 (2014). `http://arxiv.org/abs/1403.7426`