Planning and Scheduling @home project

Ángela Patricia Enríquez Gómez
Ethan Oswald Massey
Maximilian Mensing

January 18, 2019

BRSU

Stroring Groceries

- **Problem 1**: The table and the cupboard are located and known. The cupboard door is closed. One object is on the table and has to be placed on any shelf.
- **Problem 2**: The table and the cupboard have to be located. The cupboard door is closed. There are n [1] known objects on the table. The objects have to be placed on any shelf.
- **Problem 3**: As case 2 except for - The objects are not known (perception has to be used).
- **Problem 4**: As case 3 except for - Each shelf holds objects of one category (the cupboard has to be explored). Place each object on the correct shelf according to the category.

[1] n is between 2 and 5

Selection of the Planner

SHOP2 (Simple Hierarchical Ordered Planner).

- Uses partial-order forward decomposition.
- The knowledge base is easier to build than with SHOP.
- Methods allow a list of preconditions which are evaluated in order of appearance.
- Simplicity in the representation of methods and operators.

NOAH

```
(puton
  (qlambda
    (on <-X <-Y)
     (pand
         (pgoal (clear $x) (cleartop $x)
              apply (clear))
         (pgoal (clear $y) (cleartop $y)
              apply (clear))
     )
     (pgoal (put $x on top of $y)
            (on $X $y) apply nil)
     (pdeny (cleartop $y))
  )
)
```

Nonlin

```
actschema puton
  pattern <<put $*x on top of $*y>>
  conditions
    holds <<cleartop $*x>> at self
    holds <<cleartop $*y>> at self
    holds <<on $*x $*z>> at self
  effects
    + <<on $*x $*y>>
    - <<cleartop $*y>>
    - <<on $*x $*z>>
    + <<cleartop $*z>>
  vars x undef y undef z undef;
end;
```

SIPE-2

```
operator: puton
arguments: block1, object1 is not block1;
purpose: (on block1 object1);
plot:
parallel
  branch 1: goals: (clear object1);
  branch 2: goals: (clear block1);
end parallel
process
action: puton.primitive;
arguments: block1, object1;
resources: block1;
effects: (on block1 object1);
end plot end operator
```

O-Plan2

```
schema puton;
 vars ?x=undef, ?y=undef, ?z=undef;
 expands {put ?x on top of ?y};
 only_use_for_effects
    {on ?x ?y}=true,
    {cleartop ?y}=false,
    {on ?x ?z}=false,
    {cleartop ?z}=true;
 conditions
    only_use_for_query {on ?x ?y}=true,
    achievable {cleartop ?y}=true,
    achievable {cleartop ?x}=true;
endschema;
```

UMCP

```
(operator puton (x y)
   :pre ((clear x)(on x table)(clear y))
   :post ((~on x table)(on x y)(~clear y))
)
```

SHOP2

```
(:operator (!puton ?x ?y)
  ((clear ?x) (on-table ?x) (clear ?y))
  ((clear ?y) (on-table ?x))
  ((on ?x ?y))
)
```

SIADEX

```
(:action puton
 :parameters (?x ?y - block)
 :precondition (and (grasping ?x)(clear ?y))
 :effect (and (not (grasping ?x))
         (not (clear ?y))(clear ?x)
         (on ?x ?y)(handempty))
)
```

Figure 1: *put – on* operator for HTN planners [**Georgievski2014**]

Using JSHOP2

- Get jshop from: `https://github.com/mas-group/jshop2`
- Set environment: **export CLASSPATH="`pwd`/bin.build/JSHOP2.jar:`pwd`/antlr.jar:."**
- Compile: **make c**
- Run:
  - **make problem1**
  - **make problem2**
  - **make problem3**
  - **make problem4**

Modeling the domain

- Includes:
  - Operators: Directly perform primitive tasks. Use preconditions, delete and add lists.
  - Methods: Decompose compound tasks into other compound or primitive tasks. Use preconditions and task lists.
  - Axioms: Map meaning of symbols.

Problem	Compound task to achieve
1	mode-known-object ?a ?t ?c ?s
2	move-known-objects ?t ?c ?s ?tray
3	move-uncategorized-objects ?t ?c ?s ?tray ?camera
4	move-unlabeled-object-unknown-cupboard ?t ?c ?s ?tray ?camera

Table 1: Tasks to achieve for problems 1 to 4.

Assumptions

- There is exactly the number of objects required (e.g. only one table).
- The initial position of the robot is at the table.
- The robot is able to sense and identify all objects correctly.
- All objects placed on the table should be stored on the shelves.
- The robot uses a tray of infinite capacity to carry objects.
- Each shelf initially holds an object and there is one shelf per object category (problem 4).

Defining the problem

- A problem file is created for each problem.
- The problem file contains the initial state and the compound task to be solved.
- HTN planner uses the problem file and the domain to create a solution.

```
(defproblem problem1 storegroceries
;;Problem 1
(
(object a1)
(cupboard c1)
(door d1)
(shelf s1)
(table t1)
(robot r1)
(on a1 t1) (door-closed d1)(robot-at r1 t1)
)
((move-known-object a1 t1 c1 s1))
)
```

Problem 1. The table and the cupboard are located and known. The cupboard door is closed. One object is on the table and has to be placed on any shelf.

The task is decomposed using the method

move − known − object?a?t?c?s

```
1  (:method (move-known-object ?a ?t ?c ?s)
2  branch1
3  ((robot-at ?r ?t)(on ?a ?t)(door-open ?d))
4  ((!pickup ?a ?t)(!move ?r ?t ?c)(!putdown ?a ?s))
5
6  branch2
7  ((robot-at ?r ?t)(on ?a ?t)(door-closed ?d))
8  ((!move ?r ?t ?c)(!open-door ?d)(!move ?r ?c ?t)(!pickup ?a ?t)(!move ?r ?t ?c
       )(!putdown ?a ?s)))
```

GUI Result

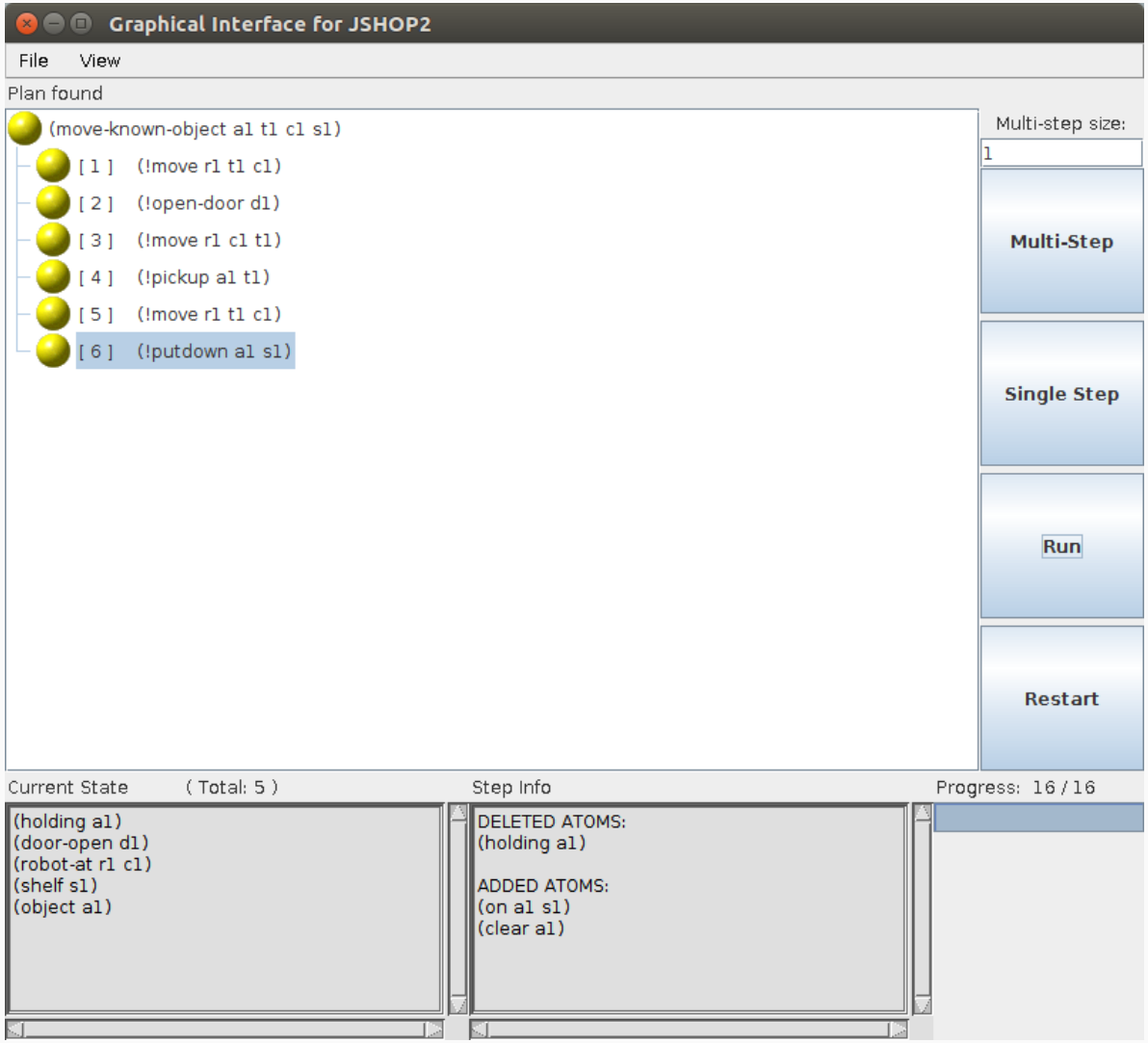

Figure 2: Solution problem 1.

Problem 4. The table and the cupboard have to be located. The cupboard door is closed. There are n [2] unknown objects on the table. Each shelf holds objects of one category (the cupboard has to be explored). Place each object on the correct shelf according to the category.

[2] n is between 2 and 5

```
(defproblem problem4 storegroceries
;;Problem4 with 2 objects
(
(object a1)
(object a2)
(camera camera1)
(cupboard c1)
(door d1)
(shelf s1)
(shelf s2)
(shelf s3)
(table t1)
(robot r1)
(tray tray1)

(unknown-location t1)(unknown-location c1)(unlabeled c1)
(holds-snack s1)
(holds-drink s2)
(holds-fruit s3)
(on a1 t1)(is-bag a1)(is-crunchy a1)
(on a2 t1)(is-bottle a2)
(door-closed d1)(robot-at r1 t1)
)
((move-unlabeled-object-unknown-cupboard t1 c1 s1 tray1 camera1))
)
```

```
 1 | [ 1 ]    (!locate t1)
 2 | [ 2 ]    (!locate c1)
 3 | [ 3 ]    (!move r1 t1 c1)
 4 | [ 4 ]    (!open-door d1)
 5 | [ 5 ]    (!label-shelf s1)
 6 | [ 6 ]    (!label-shelf s2)
 7 | [ 7 ]    (!label-shelf s3)
 8 | [ 8 ]    (!move r1 c1 t1)
 9 | [ 9 ]    (!label-object a1)
10 | [ 10 ]    (!label-object a2)
11 | [ 11 ]    (!pickup a1 t1)
12 | [ 12 ]    (!putdown a1 tray1)
13 | [ 13 ]    (!pickup a2 t1)
14 | [ 14 ]    (!putdown a2 tray1)
15 | [ 15 ]    (!move r1 t1 c1)
16 | [ 16 ]    (!pickup a1 tray1)
17 | [ 17 ]    (!putdown a1 s1)
18 | [ 18 ]    (!pickup a2 tray1)
19 | [ 19 ]    (!putdown a2 s2)
```

Table 2 compares the number of steps for generating plans for problems 2, 3 and 4 and the number of tasks in the plan, when the number of objects is 2, 3, 4 and 5.

Problem	Objects	Steps	Primitive tasks
2	2	46	14
	3	58	18
	4	70	22
	5	82	26
3	2	56	16
	3	72	21
	4	88	26
	5	104	31
4	2	70	19
	3	86	24
	4	102	29
	5	118	34

Table 2: Number of steps and tasks for plans with n objects.

Limitations

- Without sufficient prior information the planner is not able to classify objects.
- If assumptions are invalidated the planner will fail.
- In problem 4, if the shelves don't have example objects the planner does not know where to store the objects.
- Executing with java -ra generates all possible plans.
  - High space and time complexity.
  - Maximum of 4 objects to avoid outOfMemoryError.

The planner does not have information for classifying an object.

If the information needed for classifying an object is missing from the problem definition, the object will no be classified and the planner will get stuck in the recursion of the method *label − objects*.

```
1   ;; To label the objects
2   (:method (label-objects)
3   branch1
4   (forall (?z) ((object ?z))(labeled ?z))
5   nil
6
7   branch2
8   ((object ?z)(not (labeled ?z)))
9   ((!label-object ?z)(label-objects))
10  )
```

Figure 3 shows the output of the planner when 6 objects are in the problem definition and the information for classifying one of them is missing. The planner runs 17715 steps but fails to return a plan because the information about object *a*4 is missing. All objects should be labeled to stop the recursion. Since this condition is never met, the plan fails.

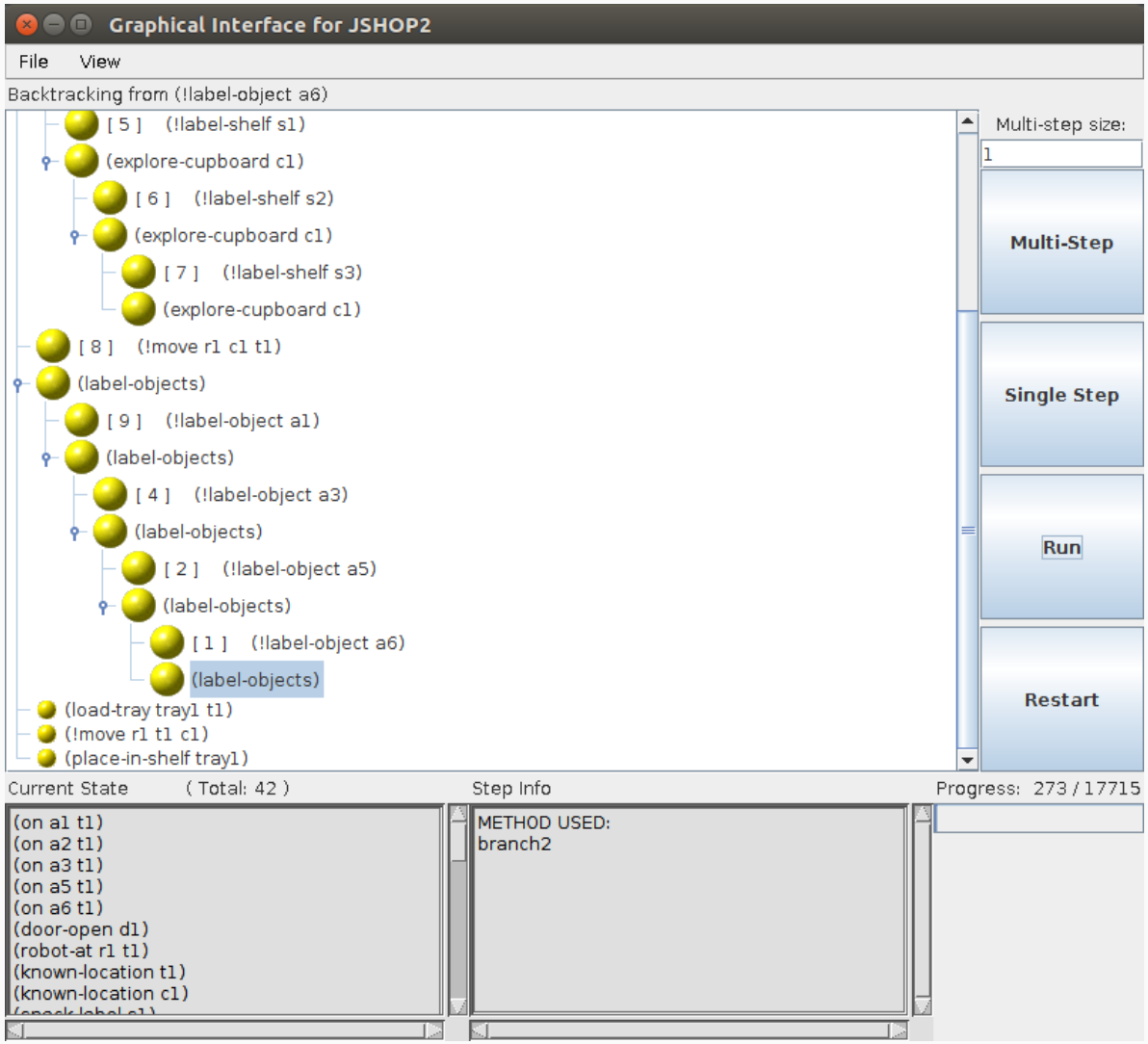

Figure 3: Failure: The planner does not have enough information for classifying an object.