A(E)3M33UI — Exercise 6:

Hierarchical planning 2

Martin Macaš

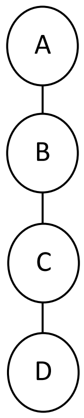
2014

Exercise 1

Before you will start to implement this exercise, find in the JSHOP2.pdf how to force JSHOP to return only the first plan found for the particular domain.

JSHOP2.pdf, page 16: Ask for all plans by `java JSHOP2.InternalDomain -ra problem,` Ask for first plan by `java JSHOP2.InternalDomain -r problem.`

Having the environment specified in the planning seminar work (graph implemented by predicate `link`), the goal is to plan a trip of soldier from one node to another. The particular environment looks like this:



Start with the problem specification using predicates `link` and `atS.` Position a soldier into node A. Define the goal node as D.

```
(defproblem demo_problem demo_domain
    (
        (link A B) (link B C) (link C D)
        (link B A) (link C B) (link D C)
        (atS Soldier A)
    )
    (
        (moveto Soldier D)
    )
)
```

Create a domain specification, which contains method `moveto ?soldier ?loc2` and operator `!GOTO ?soldier ?loc1 ?loc2` and which can be used for solving the problem defined above. The method can have three branches corresponding to the position of the soldier with respect to the goal node. (1-soldier stays at the goal node; 2-soldier stays at a neighbor of the goal node; 3-otherwise)

```
(defdomain demo_domain (
(:operator (!GOTO ?soldier ?loc1 ?loc2)
(
    (atS ?soldier ?loc1)(link ?loc1 ?loc2)
)
(
    (atS ?soldier ?loc1)
)
(
    (atS ?soldier ?loc2)
)
)

(:method (moveto ?soldier ?loc2)
branch1
(
    (atS ?soldier ?loc2)
)
()
branch2
(
    (atS ?soldier ?loc1)
    (link ?loc1 ?loc2)
)
(

    (!GOTO ?soldier ?loc1 ?loc2)

)
branch3
(
    (link ?loc1 ?loc)
)
(
    (!GOTO ?soldier ?loc1 ?loc)
    (moveto ?soldier ?loc2)
)
)
))
```

Remark 1 The label of a method branch must not contain space characters. It causes an exception.

Remark 2 Note that there is no `(atS ?soldier ?loc1)` in branch 3. It is not needed, because the planner will backtrack all the tails that do not lead to ground instances of the operators. In our case (init node is A), it is not relevant, because the planner finds the correct association at the beginning of the problem specification. However, if we initialize the soldier to node B, it still works. The association `?loc1~A ,?loc~B` is backtracked, because the condition of `!GOTO` operator is not satisfied. Question for students: why it can be useful to add `(atS ?soldier ?loc1) into branch 3?`

In the problem specification change the order of the `link` predicates and try to understand what happened:

```
(defproblem demo_problem demo_domain
     (
           (link A B)(link B A)
           (link B C)(link C B)
           (link C D)(link D C)
           (atS Soldier A)
     )
     (
          (moveto Soldier D)
     )
)
```

The problem is caused by an infinite loop. First, only precondition of branch 3 is satisfied for `?loc2~A` and `?loc~B`, thus `!GOTO soldier A B` is added and `moveto soldier  D` is called again. Again, only precondition of branch 3 is satisfied for `?loc2~B` and `?loc~A`, thus `!GOTO soldier B A` is added, `moveto soldier D` is called and the loop continues. Before, this did not happened, because of the different order of link predicates. However, if we would ask for all plans by `java JSHOP2.InternalDomain -ra problem`, the problem appears even there.

Try to modify the domain specification, which will find the plan even for this problem specification. A help can be found in the rover example.

The solution is to disable the planner to apply the branch 3 of the method to associate `?loc` with an already visited node. This can be done by addition of `(not(visited ?loc))` into the branch precondition. Then, when the soldier leaves a node, the node is marked as visited by adding predicate `visited`. After reaching the goal of *moveto*, `visited` is deleted. This can be done only by an operator. Therefore, we need two more operators - `!VISIT ?x` and `!UNVISIT ?x`.

```
(defdomain demo_domain (

(:operator (!GOTO ?soldier ?loc1 ?loc2)
(
    (atS ?soldier ?loc1)(link ?loc1 ?loc2)
)
(
    (atS ?soldier ?loc1)
)
(
    (atS ?soldier ?loc2)
)
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:operator (!VISIT ?loc)
()
()
((visited ?loc))
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:operator (!UNVISIT ?loc)
()
((visited ?loc))
()
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:method (moveto ?soldier ?loc2)
branch1
(
    (atS ?soldier ?loc2)
)
()
branch2
(
    (atS ?soldier ?loc1)
    (link ?loc1 ?loc2)
)
(
    (!GOTO ?soldier ?loc1 ?loc2)
)
branch3
(
    (link ?loc1 ?loc)
    (not(visited ?loc))
)
(
    (!GOTO ?soldier ?loc1 ?loc)
    (!VISIT ?loc1)
    (moveto ?soldier ?loc2)
    (!UNVISIT ?loc1)
)
)
))
```

Now, try to find a path from V11 to V55 through the following example environment. See what happens if you add `atS` predicate into branch 3 of the method.

```
(link V11 V21)(link V21 V11)
(link V12 V22)(link V22 V12)
(link V13 V23)(link V23 V13)
(link V14 V24)(link V24 V14)
(link V15 V25)(link V25 V15)
(link V16 V26)(link V26 V16)
(link V21 V31)(link V31 V21)
(link V22 V32)(link V32 V22)
(link V23 V33)(link V33 V23)
(link V24 V34)(link V34 V24)
(link V25 V35)(link V35 V25)
(link V26 V36)(link V36 V26)
(link V31 V41)(link V41 V31)
(link V32 V42)(link V42 V32)
(link V33 V43)(link V43 V33)
(link V34 V44)(link V44 V34)
(link V35 V45)(link V45 V35)
(link V36 V46)(link V46 V36)
(link V41 V51)(link V51 V41)
(link V42 V52)(link V52 V42)
(link V43 V53)(link V53 V43)
(link V44 V54)(link V54 V44)
(link V45 V55)(link V55 V45)
(link V46 V56)(link V56 V46)
(link V11 V12)(link V12 V11)
(link V12 V13)(link V13 V12)
(link V13 V14)(link V14 V13)
(link V14 V15)(link V15 V14)
(link V15 V16)(link V16 V15)
(link V21 V22)(link V22 V21)
(link V22 V23)(link V23 V22)
(link V23 V24)(link V24 V23)
(link V24 V25)(link V25 V24)
(link V25 V26)(link V26 V25)
(link V31 V32)(link V32 V31)
(link V32 V33)(link V33 V32)
(link V33 V34)(link V34 V33)
(link V34 V35)(link V35 V34)
(link V35 V36)(link V36 V35)
(link V41 V42)(link V42 V41)
(link V42 V43)(link V43 V42)
(link V43 V44)(link V44 V43)
(link V44 V45)(link V45 V44)
(link V45 V46)(link V46 V45)
(link V51 V52)(link V52 V51)
(link V52 V53)(link V53 V52)
(link V53 V54)(link V54 V53)
(link V54 V55)(link V55 V54)
(link V55 V56)(link V56 V55)
```