Universität des Saarlandes
Fachrichtung 6.2 – Informatik
J. Siekmann, S. Autexier, K. Fischer, E. Melis, C. Zinn
M. Schiller, E. Schulz, R.Vollmann

5th Practical Assignment in Artificial Intelligence (WS 2006/2007) **Solutions**

- **Please submit your solution by sending an email to your tutor. The e-mail address of your tutor can be found on the webpage.**
- **Indicate the group-id of your project-group.**
- **Please comment your code!**

This practical assignment should make you familiar with the planning system UCPOP

Literature:	Russell/Norvig, *Artificial Intelligence: A Modern Approach*, Chapter 11
	Barrett, Christianson, Friedman, Kwok, Golden, Penberthy, Sun, Weld, *UCPOP User's manual (Version 4.0)*, Chapters 3,4 (see web-page of the exercises)

Planning in the AIchhörnchen Domain

In this exercise you'll have to define planning operators for the UCPOP system and to execute the solution plans in the `AIchhörnchen` environment. The planning domain is the squirrel-world from the `AIchhörnchen` environment, already known from the practical exercise 2.

The environment defines a game board with fields, which may be occupied by trees and exactly one nut. There will be one squirrels, which can be moved from one field to another.
In the initial state the squirrel is in its hut and there is a nut on an arbitrary field of the game board. The goal is to put down the nut in the hut of the squirrel.

You can find information about the two systems and how to set them up in the `AIchhörnchen` manual[1] and in the UCPOP manual[2]. More information can be found on the lecture webpage[3]. If you encounter any problems you can ask for help in the lecture forum[4].
We will provide you a file which loads your solution-file, the UCPOP system and the `AIchhörnchen` environment. You can download it from `http://www.ags.uni-sb.de/~omega/teach/KI0607/material/squirrel-world.tar.gz`.

The file with the materials contains a subdirectory `gamemaster`, which again contains a subdirectory `ucpop` with the UCPOP planner.

[1]`https://xantippe.cs.uni-sb.de/~dudel/files/manual.pdf`
[2]`www.cs.colostate.edu/~anderson/cs540/labs/ucpopmanual.ps.Z`
[3]`http://www.ags.uni-sb.de/~omega/teach/KI0607/`
[4]`http://www.ags.uni-sb.de/~schiller/phpBB2`

UCPOP and AIchhörnchen

The materials contain a file `gamemaster/lisp-clients/planning.lisp`, which provides the definition of an agent using the UCPOP planner. The file `gamemaster/lisp-clients/main.lisp` is modified to load the planner. Make sure that the parameter `*ucpop-dir*` in `main.lisp` is set to the directory where the ucpop planner code (and in particular, the file `loader.lisp` from UCPOP) are located. When starting the AIchhörnchen lisp client in the usual way (i.e. `clisp -i asdf.lisp -i main.lisp -i start.lisp`), UCPOP will also be loaded.

Exercise 5.1 **(25 P)**

We use the STRIPS language to formalize the world of the AIchhörnchen (in LISP-like syntax).

- We use the atom `(field x0y1)` to express that `x0y1` is the name of a field (which has coordinates 0 and 1),
- atom `(adjacent x0y0 x0y1)` expresses that fields `x0y0` and `x0y1` are adjacent,
- atom `(at object x0y0)` expresses that an object (which can be one of `squirrel` or `nut`) is at field `x0y0`,
- and atom `(tree x0y0)` expresses that there is a tree on field `x0y0`.

The agent defined in `planning.lisp` automatically transforms the world description from the server into that representation. However, it assumes that there is only one squirrel and one nut. Then it calls UCPOP on the planning problem that consists in getting the nut and bringing it to the home base. It is your task to define the appropriate planning actions (in the syntax employed by UCPOP).

Define the preconditions and effects of the actions for each of the following actions in the squirrel-world in a file with the name `operators.lisp`:

- `move-to` (moving from one field to the next field),
- `pick-up` (picking up a nut),
- `put-down` (dropping the nut).

Check that your actions fulfil all conditions of the squirrel-world. For each correct and adequately described action you get (8 P).

Solution:

```
(defun squirrels ()
   (reset-domain)
   (define (operator move-to)
       :parameters ((field ?from) (field ?to))
       :precondition (:and (at squirrel ?from)
   (adjacent ?from ?to)
   (:not (tree ?to)))
       :effect (:and (at squirrel ?to)
     (:not (at squirrel ?from)))
       )
   (define (operator pickup-nut)
       :parameters ((field ?at))
       :precondition (:and (at squirrel ?at)
   (at nut ?at)
   (:not (has-nut))
   )
       :effect (:and (has-nut)
     (:not (at nut ?at)))
       )
   (define (operator putdown-nut)
       :parameters ()
       :precondition (:and (at squirrel x0y0)
   (has-nut)
   )
       :effect (:and (:not (has-nut))
     (at nut x0y0))
       )
)
```

Run the agent (it will take some time until UCPOP finds a plan, which is output to the screen). When a plan is found, the agent executes the steps of the plan. Hand in the successfull plan to get 1 P.

Planning in the Robot Domain

There is a robot who can move around between different rooms. Each room has a unique position (x, y). Furthermore there are objects in the room, some of which can be moved, some not.

Objects

The robots world contains two movable objects: A *Box* and a *Stick*. The movable objects are inside some rooms (i.e., movable objects have a unique position (x, y)). There can be more than one object in a room, but the robot can only carry a **single** movable object at the time.

Robot

The robot has one arm which allows it to carry a single movable object. In order to grab an object, its arm must be empty. Furthermore, it must be in the same room as the object and execute an action get-object.

If the robot moves around holding an object, then the object moves with it; i.e., the object is always in the same room as the robot is.

The robot can move between the different rooms. To simplify matters, we assume there is an operator move-to, which allows the robot to move from the current room to an arbitrary other room.

Climbing on the Box

The robot can climb on the box, if the box is in the same room than it is and it is not already standing on the box. Furthermore, it cannot climb on the box, if it is carrying the box. While the robot stands on the box, it cannot leave the room.

The lamp

In one of the rooms there is a lamp, which cannot be moved. Initially the lamp is off. Since the robot is too small to turn the lamp on while standing on the floor, it must be creative. It has the following possibilities:

- It can jump in order to turn the lamp on. But to do so, it must be in the same room as the lamp and not carrying anything.

- It can climb on the box in order to turn the lamp on. To do so the robot must be standing on the box, not carry anything and the lamp must be in the same room as the box.
- Finally, it can use the stick to reach the lamp and turn it on. To do so it must hold the stick and be in the same room as the lamp.

Planning Operators

We give you a list of actions, which the robot can take in order to achieve a given goal. These actions are given in the file `robotworld.lisp` (included in the `gamemaster/ucpop` directory in `squirrel-world.tar.gz`) which also includes additional information. You can find that file on the web-page containing the exercises. Please note the comments at the beginning of the file: they provide you with details about how to represent a state of the robot world when you define the preconditions of the actions. In this representation, rooms are given by symbolic names like `x1-y8` instead of pairs (x, y). Doing so we can concentrate on the interesting rooms, such as the room in which the robot is initially, the room where the lamp is, etc.

- `move-to`: This operator has two parameters: the room the robot is currently in and the room the robot wants to move to. To execute that action, the robot must actually be in the first room. We assume the robot can reach any room it wants to go to.
- `get-object` and `drop-object`: With these operators the robot can grab or release objects. Both have the same parameters: the object the robot wants to grab or to release and the current room. Both actions can only be executed under the conditions sketched above.
- `climb-box` and `descend-box`: With these operators the robot can climb on the box or descend from the box, if the necessary preconditions are met. These operators only have the current room as a parameter.
- `jump-light-on` `jump-light-off`: With these operators the robot can try to turn the lamp on (resp. off) by jumping. These operators only have the current room as a parameter.
- `switch-light-on` and `switch-light-off`: With these operators the robot can try to turn the lamp on (resp. off) if it is standing on the box and it is in the same room than the lamp is. These operators only have the current room as a parameter.
- `stick-light-on` and `stick-light-off`: With these operators the robot can try to turn the lamp on (resp. off) by using its stick. To do so, the robot must hold the stick and be in the same room as the lamp is. These operators only have the current room as a parameter.

Starting UCPOP

UPCOP is contained in the `squirrel-world.tar.gz` file in the `gamemaster/ucpop` directory. Set the path defined by the variable `*ucpop-dir*` in the file `loader.lisp` (in the ucpop directory) to the directory path in which you have put UCPOP (and in which loader.lisp is located).
Then start CLISP, and load the file `loader.lisp` (e.g. `(load ''loader.lisp'')`, if you started CLISP in your UCPOP directory).
Afterwards execute `(compile-ucpop)`, `(load-ucpop)` and `(in-package ''UCPOP'')`. Finally, load your version of `robotworld.lisp`. The original version of that file–which we provide you–contains the above, incomplete actions, and the following planning problems:

```
turn-on-light
stick-and-light
get-on-box
box-and-light
```

In order to generate a plan using UCPOP, you can use the function `bf-control` or its iterative deepening version `ibf-control`. Example: `(bf-control 'get-on-box)`

Exercise 5.2 **(55 P)**

Define the preconditions and effects of the actions for the robotworld domain in your copy of the file `robotworld.lisp`. Make sure to reload the file each time you changed something. Check that your actions fulfil all conditions of the robot world. Furthermore, the robot shall not execute redundant actions, like for instance to try to climb on the box if it is already standing on it.

Please comment your changes directly in your `robotworld.lisp` file. Your comments should make your lisp-code more comprehensible and explain the mode of operation of algorithms you may have inserted. For each correct and adequately described action you get (5 P).

Planning in the Robot Domain

There is a robot who can move around between different rooms. Each room has a unique position (x, y). Furthermore there are objects in the room, some of which can be moved, some not.

Objects

The robots world contains two movable objects: A *Box* and a *Stick*. The movable objects are inside some rooms (i.e., movable objects have a unique position (x, y)). There can be more than one object in a room, but the robot can only carry a **single** movable object at the time.

Robot

The robot has one arm which allows it to carry a single movable object. In order to grab an object, its arm must be empty. Furthermore, it must be in the same room as the object and execute an action get-object.

If the robot moves around holding an object, then the object moves with it; i.e., the object is always in the same room as the robot is.

The robot can move between the different rooms. To simplify matters, we assume there is an operator move-to, which allows the robot to move from the current room to an arbitrary other room.

Climbing on the Box

The robot can climb on the box, if the box is in the same room than it is and it is not already standing on the box. Furthermore, it cannot climb on the box, if it is carrying the box. While the robot stands on the box, it cannot leave the room.

The lamp

In one of the rooms there is a lamp, which cannot be moved. Initially the lamp is off. Since the robot is too small to turn the lamp on while standing on the floor, it must be creative. It has the following possibilities:

- It can jump in order to turn the lamp on. But to do so, it must be in the same room as the lamp and not carrying anything.
- It can climb on the box in order to turn the lamp on. To do so the robot must be standing on the box, not carry anything and the lamp must be in the same room as the box.
- Finally, it can use the stick to reach the lamp and turn it on. To do so it must hold the stick and be in the same room as the lamp.

Planning Operators

We give you a list of actions, which the robot can take in order to achieve a given goal. These actions are given in the file `robotworld.lisp` (included in the `gamemaster/ucpop` directory in `squirrel-world.tar.gz`) which also includes additional information. You can find that file on the web-page containing the exercises. Please note the comments at the beginning of the file: they provide you with details about how to represent a state of the robot world when you define the preconditions of the actions. In this representation, rooms are given by symbolic names like `x1-y8` instead of pairs (x, y). Doing so we can concentrate on the interesting rooms, such as the room in which the robot is initially, the room where the lamp is, etc.

- `move-to`: This operator has two parameters: the room the robot is currently in and the room the robot wants to move to. To execute that action, the robot must actually be in the first room. We assume the robot can reach any room it wants to go to.
- `get-object` and `drop-object`: With these operators the robot can grab or release objects. Both have the same parameters: the object the robot wants to grab or to release and the current room. Both actions can only be executed under the conditions sketched above.
- `climb-box` and `descend-box`: With these operators the robot can climb on the box or descend from the box, if the necessary preconditions are met. These operators only have the current room as a parameter.
- `jump-light-on` `jump-light-off`: With these operators the robot can try to turn the lamp on (resp. off) by jumping. These operators only have the current room as a parameter.
- `switch-light-on` and `switch-light-off`: With these operators the robot can try to turn the lamp on (resp. off) if it is standing on the box and it is in the same room than the lamp is. These operators only have the current room as a parameter.
- `stick-light-on` and `stick-light-off`: With these operators the robot can try to turn the lamp on (resp. off) by using its stick. To do so, the robot must hold the stick and be in the same room as the lamp is. These operators only have the current room as a parameter.

Starting UCPOP

UPCOP is contained in the `squirrel-world.tar.gz` file in the `gamemaster/ucpop` directory. Set the path defined by the variable `*ucpop-dir*` in the file `loader.lisp` (in the ucpop directory) to the directory path in which you have put UCPOP (and in which loader.lisp is located).
Then start CLISP, and load the file `loader.lisp` (e.g. `(load ‘‘loader.lisp’’)`, if you started CLISP in your UCPOP directory).
Afterwards execute `(compile-ucpop)`, `(load-ucpop)` and `(in-package ‘‘UCPOP’’)`. Finally, load your version of `robotworld.lisp`. The original version of that file–which we provide you–contains the above, incomplete actions, and the following planning problems:

```
turn-on-light
stick-and-light
get-on-box
box-and-light
```

In order to generate a plan using UCPOP, you can use the function `bf-control` or its iterative deepening version `ibf-control`. Example: `(bf-control ’get-on-box)`

Exercise P5.2: **55 P**
Define the preconditions and effects of the actions for the robotworld domain in your copy of the file `robotworld.lisp`. Make sure to reload the file each time you changed something. Check that your actions fulfil all conditions of the robot world. Furthermore, the robot shall not execute redundant actions, like for instance to try to climb on the box if it is already standing on it.

Please comment your changes directly in your `robotworld.lisp` file. Your comments should make your lisp-code more comprehensible and explain the mode of operation of algorithms you may have inserted. For each correct and adequately described action you get (5 P).

Solution:

```
(defun robot-world ()
  (reset-domain)
  (define (operator move-to)
           :parameters ((cell ?from) (cell ?to))
           :precondition (:and ;; Der Roboter mu{\ss} in dem
                               ;; angegebenen Raum sein.
                               (at robot ?from)
                               ;; Der Roboter darf nicht auf der Box
                               ;; stehen
                               (:not (on-box))
                               ;; Ziel- und Urraum duerfen
                               ;; nicht identisch sein (sonst
                               ;; w\"urde man unnoetige Schritte
                               ;; erhalten
                               (:neq ?from ?to))
           :effect (:and ;; Der Roboter ist nicht mehr im alten Raum
                         (:not (at robot ?from))
                         ;; Der Ronoter ist anschliessend im neuen Raum
                         (at robot ?to)
```

```
                          ;; Alle Objekte, die der Roboter bei sich
                          ;; traegt, sind jetzt auch im neuen Raum.
                          (:forall ((object ?obj))
                                   (:when (holding ?obj)
                                          (:and (:not (at ?obj ?from))
                                                (at ?obj ?to))))
                          )
           )
(define (operator get-object)
           :parameters ((object ?obj) (cell ?c))
           :precondition (:and ;; Der Roboter traegt nichts
                                (hand-empty)
                                ;; Der Roboter ist im Raum ?c
                                (at robot ?c)
                                ;; Das Objekt, das er aufnehmen
                                ;; soll, ist auch in ?c
                                (at ?obj ?c)
                                ;; Er darf die Box nicht aufheben, wenn er darauf steht.
                                (or (:neq ?obj box) (:not (on-box))))
           :effect (:and ;; Die Hand des Roboters ist nicht mehr frei
                          (:not (hand-empty))
                          ;; Der Roboter traegt das Objekt ?obj
                          (holding ?obj))
           )
(define (operator drop-object)
           :parameters ((object ?obj) (cell ?c))
           :precondition (:and ;; Der Roboter muss im Raum ?c sein
                                (at robot ?c)
                                ;; Der Roboter traegt Objekt ?obj
                                (holding ?obj))
           :effect (:and ;; Anschliessend ist die Hand frei
                          (hand-empty)
                          ;; und der Roboter traegt nicht mehr das
                          ;; Objekt ?obj.
                          (:not (holding ?obj)))
           )
(define (operator climb-box)
           :parameters ((cell ?c))
           :precondition (:and ;; Der Roboter ist im Raum ?c
                                (at robot ?c)
                                ;; Die Box ist auch im Raum ?c
                                (at box ?c)
                                ;; Der Roboter traegt nicht die Box
                                (:not (holding box))
                                ;; Der Roboter steht nicht schon auf
                                ;; der Box (-> Vermeidung unnoetiger
                                ;; Schritte)
                                (:not (on-box)))
           :effect ;; Der Roboter steht anschliessend auf der Box
                   (on-box)
           )
```

```
(define (operator descend-box)
           :parameters ((cell ?c))
           :precondition (:and ;; Der Roboter muss vorher auf der Box
                               ;; stehen
                               (on-box)
                               (at box ?c)
                               (at robot ?c))
           :effect ;; Anschliessend steht der Roboter nicht mehr auf
                   ;; der Box
                   (:not (on-box))
           )
(define (operator jump-light-on)
           :parameters ((cell ?c))
           :precondition (:and ;; Der Roboter muss im Raum ?c sein
                               (at robot ?c)
                               ;; Das Licht muss im Raum ?c sein
                               (at light ?c)
                               ;; Der Roboter darf nichts tragen
                               (hand-empty)
                               ;; Das Licht darf nicht schon an sein
                               (:not (light-on))
                               (:not (on-box)))
           :effect ;; Das Licht ist anschliessend an
                   (light-on)
           )
(define (operator jump-light-off)
           :parameters ((cell ?c))
           :precondition (:and ;; Der Roboter muss im Raum ?c sein
                               (at robot ?c)
                               ;; Das Licht muss im Raum ?c sein
                               (at light ?c)
                               ;; Der Roboter darf nichts tragen
                               (hand-empty)
                               ;; Das Licht darf nicht schon aus sein
                               (light-on)
                               (:not (on-box)))
           :effect ;; Das Licht ist anschliessend aus
                   (:not (light-on))
           )
(define (operator switch-light-on)
           :parameters ((cell ?c))
           :precondition (:and ;; Das Licht muss im Raum ?c sein
                               (at light ?c)
                               ;; Der Roboter muss im Raum ?c sein
                               (at robot ?c)
                               ;; Die Box muss im Raum ?c sein
                               (at box ?c)
                               ;; Der Roboter muss auf der Box sein
                               (on-box)
                               ;; Das Licht darf nicht schon an sein
                               (:not (light-on))
                               ;; Der Roboter darf nichts tragen
                               (hand-empty)
                               )
           :effect ;; Das Licht ist anschliessend an
                   (light-on)
           )
(define (operator switch-light-off)
           :parameters ((cell ?c))
           :precondition (:and ;; Das Licht muss im Raum ?c sein
                               (at light ?c)
                               ;; Der Roboter muss im Raum ?c sein
                               (at robot ?c)
                               ;; Die Box muss im Raum ?c sein
                               (at box ?c)
                               ;; Der Roboter muss auf der Box sein
                               (on-box)
                               ;; Das Licht darf nicht schon aus sein
                               (light-on)
                               ;; Der Roboter darf nichts tragen
                               (hand-empty)
                               )
           :effect ;; Das Licht ist anschliessend aus
                   (:not (light-on))
           )
(define (operator stick-light-on)
           :parameters ((cell ?c))
           :precondition (:and ;; Das Licht muss im Raum ?c sein
                               (at light ?c)
                               ;; Der Roboter muss im Raum ?c sein
                               (at robot ?c)
                               ;; Der Stock muss im Raum ?c sein
                               (at stick ?c)
                               ;; Der Roboter muss den Stock halten
                               (holding stick)
                               ;; Das Licht darf nicht schon an sein
                               (:not (light-on))
                               (:not (on-box))
                               )
           :effect ;; Das Licht ist anschliessend an
                   (light-on)
           )
(define (operator stick-light-off)
           :parameters ((cell ?c))
           :precondition (:and ;; Das Licht muss im Raum ?c sein
                               (at light ?c)
                               ;; Der Roboter muss im Raum ?c sein
                               (at robot ?c)
                               ;; Der Stock muss im Raum ?c sein
                               (at stick ?c)
                               ;; Der Roboter muss den Stock halten
                               (holding stick)
```

```
                              ;; Das Licht darf muss noch an sein
                              (light-on)
                              (:not (on-box))
                              )
           :effect ;; Das Licht ist anschliessend aus
                   (:not (light-on))
           )
  )
```

Exercise P5.3: **20 P**

Using your file `robotworld.lisp` *use UCPOP to generate plans for the following problems. There are problems where you will have to play around with the settings of the planner in order to obtain a plan. In case you nevertheless cannot solve a problem, turn in the result of the planner.*

1. `turn-on-light` *(5 P)*

Solution:

Lösung gemäß Spezifikation aus Aufgabe 1 in der Version 4.1 von UCPOP

```
turn-light-on:
==============
(bf-control 'turn-light-on)
Initial  : ((CELL X1-Y8) (CELL X5-Y1) (CELL X8-Y7) (CELL X7-Y1) (CELL X1-Y1)
            (AT ROBOT X1-Y8) (AT BOX X8-Y7) (AT STICK X1-Y1) (AT LIGHT X7-Y1)
            (OBJECT BOX) (OBJECT STICK) (HAND-EMPTY) (NOT LIGHT-ON) (NOT ON-BOX)
            (NOT HOLDING BOX) (NOT HOLDING STICK))

Step 1   : (MOVE-TO X1-Y8 X7-Y1)   Created 2
           0  -> (NOT (ON-BOX))
           0  -> (AT ROBOT X1-Y8)
           0  -> (CELL X7-Y1)
           0  -> (CELL X1-Y8)
Step 2   : (JUMP-LIGHT-ON X7-Y1)   Created 1
           0  -> (NOT (ON-BOX))
           0  -> (NOT (LIGHT-ON))
           0  -> (HAND-EMPTY)
           0  -> (AT LIGHT X7-Y1)
           2  -> (AT ROBOT X7-Y1)
           0  -> (CELL X7-Y1)

Goal     : (LIGHT-ON)
           1  -> (LIGHT-ON)
Facts:
Complete!

UCPOP Stats: Initial terms = 16;   Goals = 1 ;  Success (2 steps)
      Created 126 plans, but explored only 63
```

```
CPU time:    0.0600 sec
Branching factor:  1.984
Working Unifies: 457
Bindings Added: 139
```

---

2. `stick-and-light` *(5 P)*

Solution:

Lösung gemäß Spezifikation aus Aufgabe 1 in der Version 4.1 von UCPOP

```
stick-and-light:
================
(bf-control 'stick-and-light)
Initial  : ((CELL X1-Y8) (CELL X5-Y1) (CELL X8-Y7) (CELL X7-Y1) (CELL X1-Y1)
            (AT ROBOT X1-Y8) (AT BOX X8-Y7) (AT STICK X1-Y1) (AT LIGHT X7-Y1)
            (OBJECT BOX) (OBJECT STICK) (HAND-EMPTY) (NOT LIGHT-ON)
            (NOT ON-BOX) (NOT HOLDING BOX) (NOT HOLDING STICK))

Step 1   : (MOVE-TO X1-Y8 X7-Y1)   Created 3
           0  -> (NOT (ON-BOX))
           0  -> (AT ROBOT X1-Y8)
           0  -> (CELL X7-Y1)
           0  -> (CELL X1-Y8)
Step 2   : (JUMP-LIGHT-ON X7-Y1)   Created 4
           0  -> (NOT (ON-BOX))
           0  -> (NOT (LIGHT-ON))
           0  -> (HAND-EMPTY)
           0  -> (AT LIGHT X7-Y1)
           3  -> (AT ROBOT X7-Y1)
           0  -> (CELL X7-Y1)
Step 3   : (MOVE-TO X7-Y1 X1-Y1)   Created 2
           0  -> (NOT (ON-BOX))
           3  -> (AT ROBOT X7-Y1)
           0  -> (CELL X1-Y1)
           0  -> (CELL X7-Y1)
Step 4   : (GET-OBJECT STICK X1-Y1)   Created 1
           0  -> (AT STICK X1-Y1)
           2  -> (AT ROBOT X1-Y1)
           0  -> (HAND-EMPTY)
           0  -> (CELL X1-Y1)
           0  -> (OBJECT STICK)

Goal     : (AND (HOLDING STICK) (LIGHT-ON))
           4  -> (LIGHT-ON)
           1  -> (HOLDING STICK)
Facts:
Complete!
```

```
UCPOP Stats: Initial terms = 16;   Goals = 3 ;  Success (4 steps)
      Created 1296 plans, but explored only 694
      CPU time:    0.8800 sec
      Branching factor:  1.852
      Working Unifies: 7271
      Bindings Added: 1269
```

---

3. get-on-box *(5 P)*

Solution:

Lösung gemäß Spezifikation aus Aufgabe 1 in der Version 4.1 von UCPOP

```
get-on-box:
===========
(setf *search-limit* 5000)
(ibf-control 'get-on-box)
5348264 bytes have been tenured, next gc will be global.
See the documentation for variable EXCL:*GLOBAL-GC-BEHAVIOR* for more information.

Initial  : ((CELL X1-Y8) (CELL X5-Y1) (CELL X8-Y7) (CELL X7-Y1) (CELL X1-Y1)
            (AT ROBOT X1-Y8) (AT BOX X8-Y7) (AT STICK X1-Y1) (AT LIGHT X7-Y1)
            (OBJECT BOX) (OBJECT STICK) (HAND-EMPTY) (NOT LIGHT-ON) (NOT ON-BOX)
            (NOT HOLDING BOX) (NOT HOLDING STICK))

Step 1  : (MOVE-TO X1-Y8 X8-Y7)   Created 2
           0  -> (NOT (ON-BOX))
           0  -> (AT ROBOT X1-Y8)
           0  -> (CELL X8-Y7)
           0  -> (CELL X1-Y8)
Step 2  : (GET-OBJECT BOX X8-Y7)   Created 4
           0  -> (NOT (ON-BOX))
           0  -> (AT BOX X8-Y7)
           2  -> (AT ROBOT X8-Y7)
           0  -> (HAND-EMPTY)
           0  -> (CELL X8-Y7)
           0  -> (OBJECT BOX)
Step 3  : (MOVE-TO X8-Y7 X5-Y1)   Created 1
           4  -> (HOLDING BOX)
           0  -> (OBJECT BOX)
           0  -> (NOT (ON-BOX))
           2  -> (AT ROBOT X8-Y7)
           0  -> (CELL X5-Y1)
           0  -> (CELL X8-Y7)
Step 4  : (DROP-OBJECT BOX X5-Y1)   Created 5
           4  -> (HOLDING BOX)
           1  -> (AT ROBOT X5-Y1)
           0  -> (CELL X5-Y1)
           0  -> (OBJECT BOX)
Step 5   : (CLIMB-BOX X5-Y1)      Created 3
           0  -> (NOT (ON-BOX))
           5  -> (NOT (HOLDING BOX))
           1  -> (AT BOX X5-Y1)
           1  -> (AT ROBOT X5-Y1)
           0  -> (CELL X5-Y1)

Goal     : (AND (AT ROBOT X5-Y1) (ON-BOX))
           3  -> (ON-BOX)
           1  -> (AT ROBOT X5-Y1)
Facts:
Complete!

UCPOP Stats: Initial terms = 16;   Goals = 3 ;  Success (5 steps)
      Created 4670 plans, but explored only 2661
      CPU time:    4.6900 sec
      Branching factor:  1.716
      Working Unifies: 16331
      Bindings Added: 3345
```

---

4. `box-and-light` *(5 P)*

Solution:

Lösung gemäß Spezifikation aus Aufgabe 1 in der Version 4.1 von UCPOP

```
box-and-light:
==============
(setf *search-limit* 50000)
(ibf-control 'box-and-light)
Initial   : ((CELL X1-Y8) (CELL X5-Y1) (CELL X8-Y7) (CELL X7-Y1) (CELL X1-Y1)
            (AT ROBOT X1-Y8) (AT BOX X8-Y7) (AT STICK X1-Y1) (AT LIGHT X7-Y1)
            (OBJECT BOX) (OBJECT STICK) (HAND-EMPTY) (NOT LIGHT-ON) (NOT ON-BOX)
            (NOT HOLDING BOX) (NOT HOLDING STICK))

Step 1   : (MOVE-TO X1-Y8 X7-Y1)   Created 2
           0  -> (NOT (ON-BOX))
           0  -> (AT ROBOT X1-Y8)
           0  -> (CELL X7-Y1)
           0  -> (CELL X1-Y8)
Step 2   : (JUMP-LIGHT-ON X7-Y1)   Created 1
           0  -> (NOT (ON-BOX))
           0  -> (NOT (LIGHT-ON))
           0  -> (HAND-EMPTY)
           0  -> (AT LIGHT X7-Y1)
           2  -> (AT ROBOT X7-Y1)
           0  -> (CELL X7-Y1)
Step 3  : (MOVE-TO X7-Y1 X8-Y7)   Created 4
           0  -> (NOT (ON-BOX))
           2  -> (AT ROBOT X7-Y1)
           0  -> (CELL X8-Y7)
           0  -> (CELL X7-Y1)
Step 4  : (GET-OBJECT BOX X8-Y7)   Created 6
           0  -> (NOT (ON-BOX))
           0  -> (AT BOX X8-Y7)
           4  -> (AT ROBOT X8-Y7)
           0  -> (HAND-EMPTY)
           0  -> (CELL X8-Y7)
           0  -> (OBJECT BOX)
Step 5  : (MOVE-TO X8-Y7 X5-Y1)   Created 3
           6  -> (HOLDING BOX)
           0  -> (OBJECT BOX)
           0  -> (NOT (ON-BOX))
           4  -> (AT ROBOT X8-Y7)
           0  -> (CELL X5-Y1)
           0  -> (CELL X8-Y7)
Step 6  : (DROP-OBJECT BOX X5-Y1)   Created 7
           6  -> (HOLDING BOX)
           3  -> (AT ROBOT X5-Y1)
           0  -> (CELL X5-Y1)
           0  -> (OBJECT BOX)
Step 7  : (CLIMB-BOX X5-Y1)      Created 5
           0  -> (NOT (ON-BOX))
           7  -> (NOT (HOLDING BOX))
           3  -> (AT BOX X5-Y1)
           3  -> (AT ROBOT X5-Y1)
           0  -> (CELL X5-Y1)

Goal    : (AND (LIGHT-ON) (AT ROBOT X5-Y1) (ON-BOX))
           5  -> (ON-BOX)
           3  -> (AT ROBOT X5-Y1)
           1  -> (LIGHT-ON)
Facts:
Complete!

UCPOP Stats: Initial terms = 16;   Goals = 4 ;  Success (7 steps)
      Created 35060 plans, but explored only 20611
      CPU time:   23.5100 sec
      Branching factor:  0.000
      Working Unifies: 279823
      Bindings Added: 21865
```