FP6-004381-MACS

MACS

Multi-sensory Autonomous Cognitive Systems Interacting with Dynamic Environments for Perceiving and Using Affordances

Instrument: Specifically Targeted Research Project (STReP)

Thematic Priority: 2.3.2.4 Cognitive Systems

D2.3.2 A specification for a propositional planner and its interface to the MACS Execution Control Module

Due date of deliverable: February 28, 2007
Actual submission date v1: April 4, 2007

Start date of project: September 1, 2004 Duration: 39 months

University of Osnabrück (UOS)

Revision: Version 1

Project co-funded by the European Commission within the Sixth Framework Programme (2002–2006)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

EU Project 

Deliverable D2.3.2

A specification for a propositional planner and its interface to the MACS Execution Control Module

Christopher Lörken, Joachim Hertzberg

Number:	**MACS/2/3.2**
WP:	*2*
Status:	*revision, version 1*
Created at:	*March 7, 2007*
Revised at:	*v1 – March 30, 2007*
Internal rev:	*v14 – March 30, 2007*

FhG/AIS	Fraunhofer Institut für Intelligente Analyse- und Informationssysteme, Sankt Augustin, D
JR_DIB	Joanneum Research, Graz, A
LiU-IDA	Linköpings Universitet, Linköping, S
METU-KOVAN	Middle East Technical University, Ankara, T
OFAI	Österreichische Studiengesellschaft für Kybernetik, Vienna, A
UOS	Universität Osnabrück, Osnabrück, D

This research was partly funded by the European Commission's 6th Framework Programme IST Project MACS under contract/grant number FP6-004381. The Commission's support is gratefully acknowledged.

Corresponding author's address:

Christopher Lörken
Universität Osnabrück
Institut für Informatik
Albrechtstr. 28
D-49076 Osnabrück, Germany

Fraunhofer Institut für Intelligente Analyse- und Informationssysteme
Schloss Birlinghoven
D-53754 Sankt Augustin
Germany

Tel.: +49 (0) 2241 14-2683 (Co-ordinator)

Contact:
Dr.-Ing. Erich Rome

Joanneum Research
Institute of Digital Image Processing
Computational Perception (CAPE)
Wastiangasse 6
A-8010 Graz
Austria

Tel.: +43 (0) 316 876-1769

Contact:
Dr. Lucas Paletta

Linköpings Universitet
Dept. of Computer and Info. Science
Linköping 581 83
Sweden

Tel.: +46 13 24 26 28

Contact:
Prof. Dr. Patrick Doherty

Middle East Technical University
Dept. of Computer Engineering
Inonu Bulvari
TR-06531 Ankara
Turkey

Tel.: +90 312 210 5539

Contact:
Asst. Prof. Dr. Erol Şahin

Österreichische Studiengesellschaft für Kybernetik (ÖSGK)
Freyung 6
A-1010 Vienna
Austria

Tel.: +43 1 5336112 0

Contact:
Prof. Dr. Georg Dorffner

Universität Osnabrück
Institut für Informatik
Albrechtstr. 28
D-49076 Osnabrück
Germany

Tel.: +49 541 969 2622

Contact:
Prof. Dr. Joachim Hertzberg

Contents

1 Introduction

Currently, we are evaluating two differnet approaches to planning within the MACS architecture. The common starting point for both approaches is the (cue, behavior, outcome) affordance representation triple as defined for the affordance repository in the MACS architecture in WP2. Based on the possible interpretations of this representation, we consider the following two approaches:

- The first interpretes affordances as preconditions for plan operators. Hereby, groups of learned affordance representation triples provide the means of *grounding* the operator in terms of actually implementing the action.
- The second approach follows the idea of interpreting an affordance triple's *cue descriptor as precondition* of operators in a classical AI planning approach, and its *outcome descriptor as postcondition*. The aim is to give as little as possible domain information beforehand in order to allow the planner to make use of any learning progress that the learning module of the MACS architecture will make in affordance learning.

The first approach will be presented in chapter 2. Here, we will propose an according PDDL domain description and demonstrate the usefulness of the approach exemplary by using off-the-shelf planning systems.

The second approach will then be topic in chapter 3. This approach will show to be highly dependent on the expressive power of the output generated by the learning module. Thus, besides describing the proposed workflow, the requirements to that module will be listed.

Chapter 4 will eventually compare the benefits and drawbacks of both approaches and will address some of the questions that need still to be answered before we will give the interface definition of the planning module in chapter 5.

2 Domain description based planning

We start with adopting a fairly simple view of planning for the third year of MACS. In this view, a plan consists of a set of *operators* O and a partial order $\prec$ on these operators; if operators $o_1, o_2 \in O$ are unordered by $\prec$, they can be executed in either order. That is the classical basic view on plans as in AI. More expressive plan representations exist, but they will not be considered here at the moment.[1]

Continuing in this classical view, an operator is described by its *preconditions* and its *postconditions*; both are sets of propositional atoms, each of which is either negated or unnegated. Planning domains are typically modeled by planning domain descriptions. Amongst these, PDDL [4] is currently the most popular language to specify them and will thus be used here as well. A healthy number of algorithms and systems exists for generating plans for problems given by a PDDL domain description plus a specification of initial situation and goal propositions.

Executing a plan means to execute its operators one after another according to the plan order $\prec$. In order to execute an operator, which is first of all just a syntactic object in planning, each and every operator needs to have an implementation that tells how it should cause the robot effectors to work, and what are conditions for terminating with success, or with failure, or with timeout.

So this is classical planning as it has always been – and where do affordances come into play? Our view of the interplay between plans and affordances is this:

- Affordances help ground or implement operators and monitor plan execution.
- Affordances *may* be a precondition for some operators.

(To ground and to implement an operator will be used as synonyms here.)

So what differs from the standard view on planning if affordances can be perceived on some elementary level, is, firstly, the execution of some of the operators. An affordance and an operator are objects of distinct sorts in a robot control system. While an operator is an abstract syntactic object in planning, an affordance and its according affordance representation triples may ground an operator in the environment yielding to an executable action.

Consider, for example, the operator "pass(door1)"; and assume that an affordance "passable" is present, with the cue triggered by narrow open spaces, the behavior being to physically drive through that space, and the outcome of being located beyond the passage. One way of implementing the "pass(door1)" operator would be to make sure that the robot perceives the open door1 (i.e., it is close to it, facing it), and that it lets guide itself by the "passable" affordance of this door1 – by no other affordance that may also be present, and by this affordance as perceived in the area of door1, not door2.

So the point is: if it is clear how to act upon the perception of some affordance, then executing an operator may rely on that, provided that it can be made sure that the "right" affordance is acted upon.

[1] More precisely, O consists of different instances or occurrences of operators – "the same" operator, say, "pass(door1)" of passing door1 may occur several times in a plan. To keep things simple, we will not explicitly mention the difference between an operator class and different instances of operators of that class.

As stated above, affordances may, besides their role in the implementation of operators, also be a precondition for some operators. In order to be able to lift an object it has to be liftable. Consequently the percept of a particular affordance affords not only an action but is also a precondition for the operator to be applicable. This is the point where the planning module really exploits the MACS concept of affordances since it can, by interpreting affordances as inherent preconditional properties of an operator, determine which (in this interpretation afforded) operators to use to reach the goal. That is the second main difference of planning with or without affordances available in robot control.

See section 2.2 for the detailed description of the work flow of the planning module.

Some operators can thus be seen to be *correlated* to an affordance; to the abstract affordance and not to the instantiated affordance representation triple.

This does, however, not mean that there is a $1:1$ mapping between operators and affordances since there may be operators that cannot be mapped to an affordance. For example, part of my plan for the day may be to attend some committee meeting. Now, nothing ever affords for me to go to a committee meeting. I do it (and therefore it is part of my day plan) as a part of my professional life the reasons for which can only be given on some relatively abstract level. So to implement this operator in terms of actuator control, there simply is no affordance to exploit here.

The operator affordance relation only exists between the operator and the abstract type of the affordance. This means that the actual implementation of the operator in the world may exploit the different perceived and available affordance representation triples of that abstract affordance. In other words, the robot may have learned that both blue and red test objects are liftable. It thus has two affordance representation triples connected to the affordance liftable. In the case of the MACS project we do not distinguish test objects by their identity but by their functionality. A valid goal would therefore be to lift some liftable test object. But if this goal is described as just lifting something and not a particularly specified item, the plan will contain the lift operator if the preconditional affordance can be perceived. In this example this is the case in the presence of either red or blue cans or both. That operator may then be grounded by the execution control to the next best object that affords lifting, i.e. that fulfills the cues; even if it is completely new to the robot. In this case, the different affordance representation triples can be exploited alternatively for implementing execution of one single operator.

It is believed here, that this operator – affordance relation is meaningful exactly in those cases when an operator can be grounded by exploiting the affordance and that these are always operators that directly describe an action; like for instance, passing through a door, or lifting a can but not attending a meeting or getting coffee. Coffee affords drinking but going to the coffee machine in order to get it is just a necessary planning step before the affordance can be exploited.

Hereby, one should always keep in mind that while an operator can obviously only be correlated to at most one affordance, a test object can easily afford multiple things like, for instance, lifting, stacking, and pushing. The lift operator, for example, cannot be triggered by the affordance of stackable. Nevertheless the cues of a stackable object are most certainly a superset of the cues found in a liftable affordance representation triple. But while only those test objects that have a flat bottom can be stacked on top of others, while even round test objects may show to be liftable.

There is still another advantage that affordances offer for plan-based robot control: they make opportunistic plan execution (in the sense of [1]) possible. Classical plans normally keep track of the dependency (or causal link) structure among operators: The structure of which operator is in the plan for generating which condition for which other operator. Assume operator o is of a type that is implemented using one or many affordances. Then, depending on the concrete dependency structure, it may be possible and useful to execute o (act upon its associated affordance) before it is "its turn", namely, in case that a shortcut in the possible action at execution time is found that has emerged by serendipity or that had earlier been overlooked by incompleteness of the planning domain description. So, opportunistic plan execution may be possible, provided that affordances related to operators later in the plan can be monitored (or attended to) during plan execution.

The the following, section 2.1 will describe what kind of representation is used as the basis of the planning module to work on before section 2.2 describes the functionality that arises from this representation.

2.1 Representing the World

As aforementioned, the planning module needs knowledge about the robot's capabilities for action, i.e. what it can physically achieve, and about the current state of the world. The next section will therefore introduce the proposed map representation the robot should maintain while exploring its environment. Based on that representation, the knowledge about the capabilities for actions will be described by specifying the planner's actual operators as well as the kind of the resulting domain description in section 2.1.2.

2.1.1 Map

In order to make plan execution efficient and scalable to larger environments it is mandatory to maintain some sort of spatial representation of the environment and the objects, or test objects, within. While sophisticated techniques for mapping environments in multiple dimensions already exist, the simplified setup of the MACS demonstrator scenarios does neither demand nor justify such an elaborated approach. Nevertheless, the scenario and the resulting requirements for the functionality of the planning module are complex enough to propose a simple map as the basis of the world representation.

The kind of map proposed here is a *topological* or symbolic map exemplarily depicted in figure 1. The map depicts the two *rooms* of the demonstrator scenario and divides these rooms into a total of 6 soft *regions*; soft with the meaning of being not sharply separated. The regions are meant to be enriched with information about which (abstract) affordances have previously been perceived in that region. The idea behind this representation is to provide the robot with a most basic spatial world representation. If the robot is, for example, located in region 2 of the left room and has as its goal to drive to the right room, the resulting course of action will first make it drive to the door-region of the left room. It will drive there as it knows that to get from one room to another it has to pass through a door. And since it has previously perceived the affordance of something "passable" within or close to the door-region of the left room, it is most reasonable to drive there. When it reaches the region and perceives the affordance, the operator "drive-through" will be

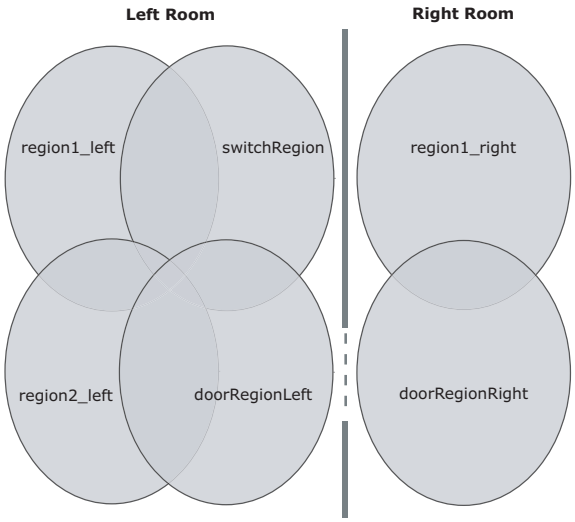

Figure 1: Topological Map - Regions are not sharply separated.

implemented in terms of an action or behavior that steers the robot through the opening, resulting in the desired goal state.

In terms of the MACS project, the map can, and maybe should, easily be predefined yielding the only requirement for the robot to localize itself coarsely to be in a particular region of a particular room. Beyond that, the robot only has to tag the map regions with the abstract affordance types it has perceived within those regions; with the *abstract* affordances and *not* with the specific triple.

At this point, the example of opportunistic planning can shortly be picked up again, as it immediately becomes clear if the robot has as to fulfill the task of lifting something. Assume that the plan sequence requests from the robot to drive to the other room in order to get to the region where it has perceived the affordance liftable. If the robot starts driving there it can on the way configure its event and execution monitor [2] in a way to be notified when the cues of the various liftable affordance representation triples are being perceived. This might for instance be the case when a new liftable object is placed in the arena by an operator while the robot is moving. The robot then perceives the according affordance before it reaches the target area and thus can abort the current action in order to implement the lift operator. It is actually beliefed that this behavior will be observable in the majority of cases since the robot will presumably perceive the test objects located in the target region, and thus the affordances connected to those objects, before it actually reaches the region.

The suggested map representation is of course only the underlying spatial representation derived from perception and learning that forms the basis for the planning module. Furthermore, it is needed to describe the knowledge encoded in that map as well as the knowledge about the applicable operators in a domain description which shall now be introduced.

2.1.2 Domain Description

A domain description forms the formal basis for a planning system. The domain normally contains sets of *operators* (see above) and furthermore *predicates* which can be evaluated to be either true or false. This is called a *domain definition*. A goal and the system's actual state is normally specified in a *problem definition* (see section 2.2). Since the current standard of specifying domain descriptions is using the Planning Domain Definition Language (PDDL) [4], it will be used here as well. We will therefore now describe a few small examples of some predicates and operators that will be contained in a similar form in the MACS planning domain description. We will do this to give an impression of the representational basis the planner will perform on and to give an idea of what quality and kind of information processing is needed by the other architectural modules to provide the necessary world representation. Section 2.2 will then continue with providing an exemplary problem definition and by explaining the work flow of the planning process.

2.1.2.1 Predicates. The predicates of a domain description are those items that can be directly evaluated; i.e. one can determine whether they are true or false in the current situation. Because of this feature, predicates are also referred to as *atoms* or *atomic formulas*. As a simple example assume:

Example 1 Predicates

```
(define (domain macs-example)
        (:requirements :strips :typing)
        (:types region room)
        (:predicates
             (robotAt ?region - region )
             (inRoom ?region - region ?room - room)))
```

This example shows the definition of 2 simple atomic formulas in PDDL syntax. The first line names the domain we are defining.[2] The second line contains the list of requirements this specification poses on the corresponding planner. Since PDDL is an abstract definition language a planner normally has to parse the description file and translate it into the corresponding planning language. The requirements in this line specify which extensions of PDDL the planner has to be able to handle to interpret the given domain description.

The part located between the surrounding brackets of the predicates section is the actual definition of the predicates. The first predicate *robotAt* is an atomic formula denoting that the robot is in a certain region. Hereby, the variable *?region* (variables are always denoted by a ?-prefix) is of type *region* that is defined in the row above (and denoted by the '- *type*' suffix). The second predicate is a formula that assigns regions to rooms. In that way one can easily build a topological map representation e.g. by specifying world facts like *(inRoom region_1 leftRoom)* or *(inRoom region_2 rightRoom)* (see section 2.2.2 for a detailed example).

[2] Assume all following listings and examples to be placed within this domain definition as well.

2.1.2.2 Operators. Given this simple definition we can go on by defining operators, or *actions* how they are called in PDDL:

Example 2 Approach Operator

```
(:requirements :strips :typing :equality)
(:action approach-region
    :parameters
        (?startRegion ?targetRegion - region
         ?room - room)
    :precondition
        (and
            (robotAt ?startRegion)
            (inRoom ?startRegion   ?room)
            (inRoom ?targetRegion  ?room)
            (not (?startRegion = ?targetRegion)))
    :effect
        (and
            (robotAt ?targetRegion)
            (not (robotAt ?startRegion)))
)
```

This example extends the defined domain by a first operator for approaching a target region in the same room. The actual operator consists of the three main parts *parameters*, *precondition*, *effect*. The list of parameters specifies the actions arguments. The precondition list contains those atomic formulas that have to hold in the current environment for the operator to be applicable while the effect list describes the changes of the world model caused by applying this operator. Note that these are the effects that are the modelled and expected outcome of the operator. If the robot fails in executing the implementation of this operator, the effect in the real world will be different to that specified here. It is, nevertheless, not task of the planning module but of perception and learning to build up and maintain a world model.

2.1.2.3 Operators and Affordances. As has been stated in the beginning of section 2, operators can also be correlated to affordances, making them a precondition for the operator. This demands to represent affordances in the domain description as well which can be achieved by formulating the predicates shown in example 3.

Example 3 Liftability Affordances

```
(:predicates
    (liftable ?region - region))
```

Here, we have introduced an abstract affordance to the planner. The truth values of this propositional formula has to be evaluated in the moment the planner is triggered either

by the user or by a replanning request. For example the world model will contain the fact *(liftable region1_left)* if the robot has previously perceived the corresponding affordance and has entered that impression in its map (see section 2.1.1). Note that only the general impression of something that somehow affords to be *liftable* is reflected by this formula but not the actual triples that hold the information whether blue or red test objects are in fact liftable. The execution module will then use all the repositories' entries of actually perceived liftable affordance representation triples to determine that environmental test object that affords to be liftable in the current situation. Again, see section 2.2 for a more detailed explanation.

Example 4 will now make use of these affordance propositions by defining a lift action that exploits the perception of liftability affordances. The following examples will furthermore show how such affordance propositions can be added or removed from the world model as the effect of an operator.

Example 4 Lift Operator with Affordance Precondition

```
(:predicates
    (hasLiftedSomething))
(:action lift-liftable
    :parameters
        (?region - region ?room - room)
    :precondition
        (and
            (robotAt ?region)
            (liftable ?region)
            (not (hasLiftedSomething)))
    :effect
        (and
            (hasLiftedSomething)
            (not (liftable ?region))))
```

This operator first of all defines an additional predicate *(hasLiftedSomething)* that reflects the state of the robot when it has something attached to its crane. As one can see, the predicates defined in the affordance definition of example 3 are being used in the precondition list of this operator. The operator will thus only be applicable by the planner, if the abstract affordance type "liftable" has been perceived at the robot's location. The outcome of the operator tells us that the system has lifted something and removes the percept of the liftable affordance for that region from the world model. Note that it might also be meaningful not to remove the liftable affordance percept from the world model as it surely might be the case that multiple liftable test objects are located in one region. Nevertheless, not removing this tag by default would result in an inexhaustible source of liftable objects which would be used by the planner if it wants to reach a goal. Thus, both options, removing the entry or leaving it in the model, are reasonable. We will stick with removing the entry for the moment. If, however, during plan execution a further liftability affordance is perceived in the region after executing the action and deleting *liftable ?region* from the world model / map, it will be re-entered directly.

Before we come to a more embracing example in section 2.2, example 5 will introduce the complementary operator for a lifting action, that is a drop operator.

Example 5 Drop Operator with Affordance Effect

```
(:action drop-liftable
    :parameters   (?region - region ?room - room)
    :precondition (and
                      (robotAt ?region)
                      (hasLiftedSomething))
    :effect       (and
                      (not (hasLiftedSomething))
                      (liftable ?region)))
```

The drop-liftable operator reverses the preconditions and effects of the lift-liftable operator. This follows, that the robot assumes to have the impression of something liftable in its region after it has dropped something that it has lifted. This allows for actually generating more complex plans as will be demonstrated in the following section when the functionality and work flow of this planning approach is described in more detail.

2.2 Functionality

The functionality and the work flow of the planning module can best be described by means of an example. We will consider a somewhat complex example to show the power of this approach.

Consider the following situation (for the map refer to figure 1, p. 5): The robot is standing in region1_left. In region2_left there is a test object that is suitable for triggering the switch and thus for opening the door. The door itself is closed as there is only standing a test object of insufficient weight on the switch. A similar test object is located in region1_right, i.e. in the other room.

As a first planning goal assume that the robot has to put the test objects that are not suitable for triggering the switch in the left room. One into region1_left and one into region2_left.

This means that the robot has to move the light test object from the switch to one of the target regions in the left room, to open the door and to get the light object in the right room to put it into the other target region.

This more complex setup now demands for a more powerful domain description. We will need predicates that represent all the relations and facts just mentioned. We furthermore will need a propositional equivalent telling the planner about perceived affordances and finally we will need a set of operators allowing to trigger the switch, lifting and dropping different kinds of objects as well as for moving around in the environment.

A domain description for this scenario will thus now be provided followed by a problem definition in section 2.2.2 and the description of the planning process and result in section 2.2.3.

2.2.1 Domain Description

We will start again with some predicates that represent world facts or states of the robot[3] (see predicates listing 1).

Predicates 1 Current State Predicates

```
(inRoom ?region - region ?room - room)
(robotAt ?robotRegion - region )
(hasLiftedSomething)
(hasSwitchReleaserLifted)
```

The first three predicates should already be familiar from example 4. But moreover, we have extended the list of predicates by defining a state predicate that represents the fact that the robot has lifted something that may be used for triggering the switch.

This directly leads to those predicates that map the robot's perceived affordances to planable propositions. To be able to solve the introduced task, we define affordance predicates representing the affordance percepts of something liftable, something passable, and moreover some affordances connected to the interaction possibilities of the switch:

Predicates 2 Affordance Proposition Predicates

```
(non-releaser-liftable ?region - region)
(passable ?startRegion - region ?targetRegion - region)
(switch-releaser-liftable ?region - region)
(switch-triggerable ?region - switchRegion )
(affords-removing-from-switch ?region - switchRegion )
```

Here we can already see that we distinguish two affordances for liftable objects. There are two groups of test objects amongst the objects that are perceived as being liftable; the group of test objects that can be used to trigger the switch and the group of those that do not have the necessary properties.

It is important to note that we do not at all define what properties are necessary for categorizing an object to belong to one or the other group. The planner does not care about this information. It is assumed that the learning module will generate affordance representation triples based on the observation of the robot's own actions. If the robot succeeded in triggering the switch, e.g., with a blue test object one of these affordance representation triples will contain a *blue* representation in its cue descriptor. As the system succeeded in triggering the switch with this object, the affordance triple will have the type *switch-releaser-liftable*. This clearly distinguishes this test object, e.g., from those red objects that can be lifted but do not trigger the switch. The information that is encoded in the actual triple's cue descriptor is, however, completely learned and not predefined. The planner will work only on a representation telling in which region of the map the abstract switch-releaser-liftable affordance (*not* a specific triple) has previously been perceived.

[3]The full domain and problem specification can be found in the appendix.

Furthermore, we have introduced predicates that signal the perception of a *passable* affordance, coding in this context for the affordance to drive through the open door, as well as an affordance suggesting to trigger the switch and one suggesting to remove an object from the switch. The following operators demonstrate how these affordance propositions are used in the modeling of corresponding operators and show in the following how the planner can thus exploit its knowledge about affordances and action to generate "affordance-based" plans.

Besides the approach operator of example 2 (p. 7), we will now introduce a set of operators that use this knowledge and exploit it for generating appropriate plans.

The operators 1 and 2 define two different lift operators that distinguish themselves most importantly in the preconditional affordance. While the first operator describes an action that lifts a test object that affords to be liftable but does not afford to serve as a switch triggering weight, the second operator describes the action of lifting a test object affording exactly this triggering process. The important part is, that the Lift-Switch-Releaser operator defines the additional output of *(hasSwitchReleaserLifted)* adding this fact to the world model of the robot. Nevertheless, both operators have the effect of having lifted something. The planner will exploit this knowledge when it comes to operator selection.

Operator 1 Lift-Non-Releaser

```
(:action lift-non-releaser
    :parameters (?region - region )
    :precondition
        (and
            (robotAt ?region )
            (non-releaser-liftable ?region )
            (not (hasLiftedSomething)))
    :effect
        (and
            (hasLiftedSomething)
            (not (non-releaser-liftable ?region))))
```

The two complementary operators that drop the lifted objects are specified in operator 3 and 4. These two operators turn around the precondition and effect clauses of the lift operators. Note that this results in having the affordance that triggered the lift operator in the effect list of the drop operators.

These first four operators allow the robot to lift and to drop liftable test objects and to distinguish whether these objects have been perceived to afford triggering the switch or not. If the robot no has lifted a test object that affords this action, i.e. if the world model contains *(hasSwitchReleaserLifted)*, the robot may trigger the switch if it perceives the *switch-triggerable* affordance (see operator 5).

The *trigger-switch* operator describes the preconditions and effects of putting a switch-releaser test object on the switch. The effect contains that the passage between the two door regions of the map will be *passable* in both directions. The switch will not be triggerable any more but the object-on-switch setup will yield the impression that

Operator 2 Lift-Switch-Releaser

```
(:action lift-switch-releaser
    :parameters (?region - region ?room - room)
    :precondition
        (and
            (robotAt ?region)
            (switch-releaser-liftable ?region)
            (not (hasLiftedSomething)))
    :effect
        (and
            (hasSwitchReleaserLifted)
            (hasLiftedSomething)
            (not (switch-releaser-liftable ?region))))
```

Operator 3 Drop-Non-Releaser

```
(:action drop-non-releaser
    :parameters (?region - region)
    :precondition
        (and
            (robotAt ?region)
            (hasLiftedSomething)
            (not (hasSwitchReleaserLifted)))
    :effect
        (and
            (not (hasLiftedSomething))
            (non-releaser-liftable ?region)))
```

Operator 4 Drop-Switch-Releaser

```
(:action drop-switch-releaser
    :parameters (?region - region)
    :precondition
        (and
            (robotAt ?region)
            (hasSwitchReleaserLifted))
    :effect    (and
            (not (hasSwitchReleaserLifted))
            (not (hasLiftedSomething))
            (switch-releaser-liftable ?region)))
```

Operator 5 Trigger-Switch

```
(:action trigger-switch
    :parameters
            (?doorRegion ?otherDoorRegion - doorRegion
             ?switchRegion - switchRegion)
    :precondition
        (and
            (robotAt ?switchRegion)
            (hasSwitchReleaserLifted)
            (switch-triggerable ?switchRegion)
            (not (= ?doorRegion ?otherDoorRegion)))
    :effect
        (and
            (passable ?doorRegion ?otherDoorRegion )
            (passable ?otherDoorRegion ?doorRegion )
            (not (switch-triggerable ?switchRegion))
            (affords-removing-from-switch ?switchRegion)
            (not (hasSwitchReleaserLifted))
            (not (hasLiftedSomething))))
```

something can be removed from the switch (see operator 6). Note that the case of putting a non-releaser object on the switch is skipped here for reasons of brevity. See appendix A for the complete list of operators.

Nevertheless, by defining the *remove-releaser-from-switch* and *remove-non-releaser-from-switch* operators, the planner will be able to exploit the affordance of something that affords removing (operators 6 and 7). In the case of these two operators, the planner can choose the correct one based on the perception of a passable affordance. If the passage is available, the item on the switch must have triggered the switch. If the affordance of something passable cannot be perceived, the item did not have the characteristics of a switch-releaser and thus does not result in the *(hasSwitchReleaserLifted)* effect.

The last operator that needs to be defined to complete this fairly complex domain representation is the operator that allows the robot or the planner respectively to change from one room to another (see operator 8). This operator exploits the aforementioned *passable* affordance to move from one door region to the other.

2.2.2 Problem Definition

Given the domain description of the last section we need to specify a problem definition that contains the current state of the environment and furthermore a goal. The planner tries to achieve this goal by constructing an operator sequence that will lead to the desired goal if implemented as foreseen by the execution control. The problem definition that describes the setup described at the beginning of section 2.2 is given in the problem definition listing 1.

Operator 6 Remove-Releaser-From-Switch

```
(:action remove-releaser-from-switch
    :parameters
            (?doorRegion ?otherDoorRegion - doorRegion
             ?switchRegion - switchRegion)
    :precondition
        (and
            (robotAt ?switchRegion)
            (not (= ?doorRegion ?otherDoorRegion))
            (not (hasLiftedSomething))
            (affords-removing-from-switch ?switchRegion)
            (passable ?doorRegion ?otherDoorRegion)
            (passable ?otherDoorRegion ?doorRegion))
    :effect
        (and
            (not (passable ?doorRegion ?otherDoorRegion))
            (not (passable ?otherDoorRegion ?doorRegion))
            (switch-triggerable ?switchRegion)
            (not (affords-removing-from-switch ?switchRegion))
            (hasSwitchReleaserLifted)
            (hasLiftedSomething)))
```

Operator 7 Remove-Non-Releaser-From-Switch

```
(:action remove-non-releaser-from-switch
    :parameters
            (?doorRegion ?otherDoorRegion - doorRegion
             ?switchRegion - switchRegion)
    :precondition
        (and
            (robotAt ?switchRegion)
            (not (hasLiftedSomething))
            (not (= ?doorRegion ?otherDoorRegion))
            (affords-removing-from-switch ?switchRegion)
            (not (passable ?doorRegion ?otherDoorRegion))
            (not (passable ?otherDoorRegion ?doorRegion)))
    :effect
        (and
            (switch-triggerable ?switchRegion)
            (not (affords-removing-from-switch ?switchRegion))
            (hasLiftedSomething)))
```

Operator 8 Change-Room

```
(:action change-room
    :parameters
            (?doorRegion - doorRegion
             ?targetDoorRegion - doorRegion)
    :precondition
        (and
            (robotAt ?doorRegion)
            (not (= ?doorRegion ?targetDoorRegion))
            (passable ?doorRegion ?targetDoorRegion))
    :effect
        (and
            (not (robotAt ?doorRegion))
            (robotAt ?targetDoorRegion)))
```

Problem Definition 1 Problem Definition

```
(define (problem macs-prob)
    (:domain macs-example)
    (:objects
        region1_left region2_left region1_right - region
        switchRegion - switchRegion
        doorRegionLeft doorRegionRight - doorRegion
        rightRoom - room
        leftRoom - switchRoom)
    (:init
        (robotAt region1_left)
        (inRoom region1_left leftRoom)
        (inRoom region2_left leftRoom)
        (inRoom switchRegion leftRoom)
        (inRoom doorRegionLeft leftRoom)
        (inRoom region1_right rightRoom)
        (inRoom doorRegionRight rightRoom)

        (switch-releaser-liftable region2_left)
        (non-releaser-liftable region1_right)
        (affords-removing-from-switch switchRegion))
    (:goal
        (and (non-releaser-liftable region1_left)
            (non-releaser-liftable switchRegion)
            (switch-triggerable switchRegion)
            (not (hasLiftedSomething)))))
```

The problem definition contains first a link to the domain definition that defines the different predicates used here. The *objects* parentheses include all different objects that the planner can work with. Remember that we leave the task of grounding the operators in the environment to the execution control that is meant to exploit the learned affordance representation triples. That is why the object list in this representation only encodes the map and its rooms and regions as it was introduced in section 2.1.1.

The *init* parentheses surround the initial state of the system that is meant to reflect the environment in the moment in which the planner is triggered. The first block reflects the static environment while the second block reflects those propositions that emerge from the current knowledge of the world. In this case, it is assumed that the robot has perceived the affordance of switch-releaser-liftable in region2_left, non-releaser-liftable in region1_right, and furthermore that in the switch region the action of removing something from the switch is perceived as being afforded. This initial state reflects the given description of the scenario (see the beginning of section 2.2.

The last block describes in the goals of the system that demands for the robot to get the impression of something liftable that does, however, not afford to be lifted in two different regions of the left room. Furthermore the switch shall be perceived as affording to be triggered and the robot shall not hold anything.

2.2.3 Planning Process

Based on the PDDL domain definition and problem description, we can use an off-the-shelf planning system for delivering a valid plan. In this case, we use the Fast Forward (FF) planner of Jörg Hoffmann [3][4]. Listing 1 shows the plan as it was generated by the FF planner.

This plan is a sequence of operators that transform the initial situation description into another one in which the goals are true (i.e., contained). The plan contains all necessary steps to open the door and to bring about the desired affordance percepts within the specified target regions. If the robot succeeds in executing the different steps of the plan, the resulting state of the world will correspond to the world model as specified in the goal description of the planning problem. If, however, the execution of one of these operators fails, the planner has to be triggered again, with an updated world model in order to deliver a new plan corresponding to the current and changed situation.

[4]This GPL planner can be obtained at `http://members.deri.at/~joergh/ff.html`.

Plan 1 FF Generated Plan

0:	APPROACH-REGION	region1_left	switchregion	leftroom
1:	REMOVE-NON-RELEASER-FROM-SWITCH			
		doorregionright	doorregionleft	switchregion
2:	APPROACH-REGION	switchregion	region1_left	leftroom
3:	DROP-NON-RELEASER	region1_left		
4:	APPROACH-REGION	region1_left	region2_left	leftroom
5:	LIFT-SWITCH-RELEASER	region2_left	leftroom	
6:	APPROACH-REGION	region2_left	switchregion	leftroom
7:	TRIGGER-SWITCH	doorregionright	doorregionleft	switchregion
8:	APPROACH-REGION	switchregion	doorregionleft	leftroom
9:	CHANGE-ROOM	doorregionleft	doorregionright	
10:	APPROACH-REGION	doorregionright	region1_right	rightroom
11:	LIFT-NON-RELEASER	region1_right		
12:	APPROACH-REGION	region1_right	doorregionright	rightroom
13:	CHANGE-ROOM	doorregionright	doorregionleft	
14:	APPROACH-REGION	doorregionleft	switchregion	leftroom
15:	DROP-NON-RELEASER	switchregion		
16:	REMOVE-RELEASER-FROM-SWITCH			
		doorregionright	doorregionleft	switchregion
17:	DROP-SWITCH-RELEASER	switchregion		

3 Cue – Outcome based planning

The second approach towards an affordance-based or -inspired planning module aims at incorporating the actual affordance representation triples, and not only their abstract types, more deeply into the planning process. The main idea can shortly be described as planning on the *cue* and *outcome descriptors* of the triples. More precisely, it is assumed that the robot has learned, in form of the affordance representation triples, which cues are necessary for an action to be applicable (or to be afforded) and what are the results of applying that action in the corresponding situation.

If the outcome and cue descriptions can now be mapped onto each other in a way that they can be unified, the system may be able to sequentialize different affordance representation triples, that hold the information for an action's execution, into a plan. This plan thus contains the information of how to transform an initial situation in a goal-directed manner into a different situation that contains the user-specified goals as part of the last triple's outcome descriptor.

In other words: If the robot has learned, that a red blob cue triggers the affordance *liftable*, and it knows that such a red blob can be perceived if the *open-door* affordance is being used and implemented, a plan can be generated that tells the system to act upon the open-door affordance representation triple and afterwards on the red blob liftable triple. At this moment, it is, nevertheless still not fully specified how the actual planning process can work on this representation. The cue and outcome descriptors of the affordance representation triples have to serve as the precondition and effect part of a formal operator description to give a planning system a representation to work on. What is clear up to now

is that the cue and outcome descriptors of the affordance representation triples somehow have to be unifiable. It has to be possible to map the representation of an action's outcome to the cues that can trigger another affordance. This and other remaining open questions will be addressed in section 4.3.

4 Comparison of the two approaches

Both approaches introduced in the last two chapters have their benefits and drawbacks. This chapter provides a brief overview of their different characteristics.

4.1 Benefits and drawbacks of the fully specified approach

+ There exist virtually no requirements on the actual structure of an affordance representation triple's cue and outcome descriptors. The planner does not need to evaluate the information encoded within the specific triples as it only plans using abstract affordance types. The different descriptors can therefore hold all the information the learning module or the execution control and its Event and Execution Monitor can handle. This explicitly allows for storing more complex time series of temporal changes within the descriptors, e.g. in the form of complex MITL formulas.

+ Symbol, or operator, grounding is an important research question and can still be regarded as being unsolved today. Using learned affordance representation triples to implement or ground the operators delivered by the planner on environmental test objects that afford the corresponding action yields a most interesting and valuable contribution to this field of research.

- The system uses explicitly predefined world knowledge that encodes information about the possible exploits of affordances and the connections of affordances to actions and their complements. Such an approach is, firstly, hard to extend, for instance, if new actions should be learned as this would need newly defined operators. Secondly, when using a world representation and domain description developed purely based on expert knowledge, the system will lack an important part of autonomy and does not fully exploit the power of current learning approaches.

4.2 Benefits and drawbacks of cue and outcome based planning

+ The approach minimizes the need of predefined operators and thus the usage of expert knowledge. Most relations and the characteristics of affordances are learned and the learned knowledge is explicitly used for planning. The system thus becomes highly flexible and scalable. For instance, the learning of new actions is theoretically easy as each affordance representation triple can already be regarded as encoding its own, meaningful action.

- The approach has high demands on the representation of the cue and outcome descriptors. First of all, they have to be unifiable to be able to generate plans by deriving world facts that are, on the one hand, the outcome of an action and affordance representation triple respectively, and that lead, on the other hand, to perceiving the cue of another triple for being able to build a sequential plan of triples. Secondly, the descriptors have to encode their information on a symbolic level as, e.g., complex time series of value development cannot easily be mapped onto each other. Especially since temporal planning systems cannot be integrated into the project in the short remaining period of time and it is not yet clear if or how, for instance, an

EEM capable MITL representation of a cue can be translated into a valid symbol usable by a propositional planner.

- The planning procedure and execution becomes much harder. While the first approach works on a cleanly defined domain description, this approach aims at planning mostly on learned data. The search space will presumably be significantly larger than in the other approach. Plan execution is more likely to fail as a plan would contain only one affordance representation triple for fulfilling a step instead of all those triples that belong to the corresponding abstract affordance category. Due to this circumstance, plans will as well be less flexible and failing to detect the cues of a planned triple will eventually result in a failed plan execution and thus in the need for subsequent replanning.

$\sim$ It is not yet clear how to include facts like the need of driving to a target location before following the affordance of dropping something can be encoded within the plans and the whole approach. Modeling movement as affordance representation triples is certainly not a good idea.

4.3 Open Questions

General questions:

- Who can build and maintain a map of the robot's environment and how should this procedure work exactly?

Second approach:

- How can cues and outcomes be represented to serve both as input to the EEM and the planner as well?
- How can a valid mapping between cues and outcomes be achieved?
- How can we account for space differences within plans of affordance representation triples?
- Can, and if yes how can, a map representation be integrated in the second approach?

4.4 Conclusion

Both approaches presented above have strong benefits and are suitable for approaching into the direction of fulfilling the MACS goal of evaluating the possible applicability and usefulness of affordance theory to robotics.

At first sight, the second approach seems to be more powerful since it is definitely more scalable works completely on learned data while it does not at all depend on expert knowledge. The first approach, nevertheless, shows benefits in exploiting the flexibility of affordances by leaving the grounding of operators or actions completely to the learned interaction possibilities offered by the environment tackling in fact one of the main points of affordance theory.

As the cue and affordance based planning approach poses more requirements to the architectural integration and especially to the affordance representation triples' explicit

content (which is not yet defined) it is more feasible to start with the other approach that is less demanding in this regard. Chapter 2 already gives a usable domain definition and demonstrates the approach's applicability with a standard planner yielding a correctly developed and executable plan. Due to these considerations work has started to define interfaces with the Execution Module and we decided to start off with the domain description based planning approach for achieving a first demonstratable prototype system.

During the development of this system the content of the affordance representation triples' descriptors will be defined in more detail and work can start towards defining and implementing the second approach of cue - outcome based planning as well.

5 Interface to the Execution Control Module

This section describes the interface between the Planning Module (PM) and the Execution Module (EM).[5] As a general note clarifying the communication between these modules we have decided to use the PM solely for delivering plans that are sequences of operators. The EM will contain a scheduling component implementing each plan operator step-by-step by using its Event and Execution Monitor as well as the Behavior Module (BM).

Though the interface between PM and EM will be held in a way abstract enough to support partially ordered plans, we will begin with totally ordered plans in the first demonstration setups.

Generally, it is important to keep in mind that there exists a mapping from operators to behaviors but that they are not at all in a 1:1 relation. The next section will thus give a detailed list of what behaviors the EM will use to execute the operators before section 5.2 will eventually give the interface definition between PM and EM.

5.1 List of Operators and their EM counterparts

The following table will provide the (not yet complete) list of operators specified for the domain description based planning approach as well as their Behavior Module counterparts which shall be instantiated by the EM in order to implement the operators. Remember that the test objects that will be used to ground them will be selected by the EM as well by using the EEM.

Operator	**BM Counterpart**	**Remark**
approach-region	Approach	
lift-non-releaser	Lift	
lift-switch-releaser	Lift	
drop-non-releaser	Drop	
drop-switch-releaser	Drop	
trigger-switch	Stack	
remove-releaser-from-switch	Unstack	
remove-non-releaser-from-switch	Unstack	
change-room	DriveThrough	
carry	Approach	monitor lifted weight
carry-through	DriveThrough	monitor lifted weight
...		

(Note that the operators needed for the stacking demonstrator scenario are not yet specified.)

The *carry* and *carry-through* operators are special as they signal a special constraint to the EM to monitor during their execution. The EM has to monitor that the weight attached to the crane does not vanish during the execution of the action. If it does, the EM can detect the failure and react appropriately.[6]

[5] Note that this interface definition is not yet fully specified and is thus subject to change.

[6] Note that these two operators were, for reasons of simplicitly, not included in the examples presented in section 2.

5.2 Interface definition

The interface of the Planning Module is defined as follows:

```
interface IPlanningModule {

   typedef sequence<ART> AffordanceTripleVector;
   typedef sequence<string> ParameterVector;

   struct StrOperator {
     string name;
     AffordanceTripleVector affordanceTriples;
     ParameterVector parameters;
   }

   typedef sequence<StrOperator> Plan;
   typedef sequence<string> Goal;

   plan(in Goal goal, out Plan plan);
   replan(out Plan plan);
}
```

Hereby, *ART* is a placeholder for the affordance representation triple structure that is not yet defined as it depends on the Learning Module. An operator consists of an identifying name, a sequence of these ARTs and a sequence of parameters as well. The AffordanceTrippleVector will, in the case of the domain description based planning approach (section 2), contain all those learned affordance representation triples that belong to the type of affordance associated with the operator; e.g., blue blob is cue for liftability, red blob is cue for liftability.

However, not all operators need an ART to be applicable and executable by the EM. For instance approach-region and carry do not need an affordance. These operators will simply have an empty AffordanceTripleVector but instead specify necessary parameters by filling the ParameterVector. An example would be the identifying name of the region that the robot is meant to approach.

Of course, the Planning Module interface defines plans and the necessary planning functions as well. A plan (in this case still totally ordered, partially will probably follow) is a sequence of planning operators. A plan is achieved by calling the plan() function that requires an input in form of a goal description and delivers as output an according plan. If the planner cannot find a valid plan to reach the goal, the resulting plan will be empty.

The replan() function uses the same goal that was given to the plan() function beforehand but tries to deliver a new plan based on the updated world state.

The EM will provide the following interface to execute plans or to stop the execution.

```
interface IExecutionModule {

   executePlan(in Plan plan);
   stopExecution();
}
```

While the PM's replanning process will probably mainly be triggered from the Execution Module, the EM's execution and stopExecution as well as the PM's initial planning function will mainly be called from the user interface.

References

[1] B. Hayes-Roth and F. Hayes-Roth. A cognitive model of planning. *Cogn. Sci.*, 3:275–310, 1979.

[2] F. Heintz, P. Doherty, B. Wingmann, P. Rudol, and M. Wzorek. A software prototype for an affordance monitoring module with empirical testing using various MACS robotics platforms – The Event and Execution Module (EEM). Deliverable D4.4.1, MACS Internal Technical Report, Linköpings Universitet (LiU/IDA), Linköping, Sweden, October 2006. Draft, version 2.

[3] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[4] D. McDermott. PDDL - The planning domain definition language. Technical report, Yale University, 1998.

A domain.pddl

```
;; Domain MACS Example
;; PDDL definition of the MACS domain and the operators needed
;; for changing rooms and opening the door by triggering the
;; switch with appropriate test objects.
;; Assumption is that the operators are fully specified and that
;; a map representation holds the abstract affordance types
;; represented in regions.

(define (domain macs-example)


   (:requirements :strips :typing :equality )


   (:types doorRegion switchRegion - region
           switchRoom - room )


; ########### Predicates ################
   (:predicates

        ; AFFORDANCE PROPOSITIONS:
        ; affordance propositions get an assigend truth value
        ; determined from the snapshot of the world in the moment
        ; in which the planner is triggered.
        ; They are as well encoded in the effect part of an operator
        ; that results in this affordance to be available.
        ; E.g.:
        ;     - drop action follows liftable, or
        ;     - trigger switch follows passable.

        (non-releaser-liftable ?region - region )
        (switch-releaser-liftable ?region - region )

        ; open passage from one room to another.
        ; Implies knowledge of a map, and knowledge of the effect of
        ; opening doors, etc.
        (passable
                ?startRegion - region
                ?targetRegion - region )
        ; true if the switch is empty so one can place something on it
        (switch-triggerable ?region - switchRegion )
        ; true if something can be removed from the switch
        (affords-removing-from-switch ?region - switchRegion )


        ; NORMAL PREDICATES:

        ; The robot is in a certain region of a certain room.
        ; Note that we do not have to encode explicit entities as we
```

```
        ; only consider abstract affordances that have been perceived
        ; in the corresponding regions.
        (robotAt ?robotRegion - region )
        ; true if a region is in a room
        (inRoom ?region - region ?room - room )
        ; true if the robot has lifted something
        (hasLiftedSomething )
        ; true if robot has lifted an object that affords
        ; releasing the switch
        (hasSwitchReleaserLifted )


) ;; End predicates



; ##########   Actions  ################

(:action approach-region
    :parameters (?startRegion - region
                 ?targetRegion - region
                 ?room - room)
    :precondition
        (and
            (robotAt ?startRegion )
            ; make sure start and target region are in the same room
            (inRoom ?startRegion ?room )
            (inRoom ?targetRegion ?room )
            (not (= ?startRegion ?targetRegion )))
    :effect
        (and
            (robotAt ?targetRegion )
            (not (robotAt ?startRegion ))))


(:action lift-non-releaser
    :parameters (?region - region )
    :precondition
        (and
            (robotAt ?region )
            ; liftable perceived in that region?
            (non-releaser-liftable ?region )
            (not (hasLiftedSomething )))
    :effect
        (and
            (hasLiftedSomething )
            (not (non-releaser-liftable ?region ))))


(:action lift-switch-releaser
    :parameters (?region - region ?room - room )
    :precondition
        (and
```

```
            (robotAt ?region )
            (switch-releaser-liftable ?region )
            (not (hasLiftedSomething )))
    :effect
        (and
            (hasSwitchReleaserLifted )
            (hasLiftedSomething )
            ; negate the switch-releaser-liftable affordance for this
            ; location. If we would not do this, the
            ; planner would have an inexhaustible source
            ; of releaser liftable test objects here.
            (not (switch-releaser-liftable ?region ))))


(:action drop-non-releaser
    :parameters (?region - region )
    :precondition
        (and
            (robotAt ?region )
            (hasLiftedSomething )
            ; the switch releaser has an own drop operator
            (not (hasSwitchReleaserLifted )))
    :effect
        (and
            (not (hasLiftedSomething ))
            (non-releaser-liftable ?region )))


(:action drop-switch-releaser
    :parameters (?region - region )
    :precondition
        (and
            (robotAt ?region )
            (hasSwitchReleaserLifted ))
    :effect
        (and
            (not (hasSwitchReleaserLifted ))
            (not (hasLiftedSomething ))
            ; switch-releaser liftable affordance will be perceivable
            ; in this region after dropping.
            (switch-releaser-liftable ?region )))


(:action trigger-switch
    :parameters   (?doorRegion ?otherDoorRegion - doorRegion
                   ?switchRegion - switchRegion )
    :precondition
        (and
            (robotAt  ?switchRegion )
            (hasSwitchReleaserLifted )
            ; Does the switch afford to be triggered
            ; (i.e.: is it empty?)
            (switch-triggerable ?switchRegion )
```

```
            (not (= ?doorRegion ?otherDoorRegion )))
    :effect
        (and
            ; result is affordance perception of something passable
            (passable ?doorRegion ?otherDoorRegion )
            (passable ?otherDoorRegion ?doorRegion )
            ; the switch will not afford to be triggered again
            (not (switch-triggerable ?switchRegion ))
            ; it will afford to remove something from it
            (affords-removing-from-switch ?switchRegion)
            (not (hasSwitchReleaserLifted ))
            (not (hasLiftedSomething ))))


(:action remove-releaser-from-switch
    :parameters   (?doorRegion ?otherDoorRegion - doorRegion
                   ?switchRegion - switchRegion )
    :precondition
        (and
            (robotAt  ?switchRegion )
            (not (= ?doorRegion ?otherDoorRegion ))
            (not (hasLiftedSomething ))
            (affords-removing-from-switch ?switchRegion )
            ; remove heavy implies that the passage can be perceived.
            (passable ?doorRegion ?otherDoorRegion )
            (passable ?otherDoorRegion ?doorRegion ))
    :effect
        (and
            ; result is door == closed == not passable
            (not (passable ?doorRegion ?otherDoorRegion ))
            (not (passable ?otherDoorRegion ?doorRegion ))
            ; the switch will afford to be triggered
            (switch-triggerable ?switchRegion )
            ; it will not afford to remove something from it anymore
            (not (affords-removing-from-switch ?switchRegion ))
            ; the door was open --> we have removed something heavy
            (hasSwitchReleaserLifted )
            (hasLiftedSomething )))


(:action remove-non-releaser-from-switch
    :parameters   (?doorRegion ?otherDoorRegion - doorRegion
                   ?switchRegion - switchRegion )
    :precondition
        (and
            (robotAt  ?switchRegion )
            (not (hasLiftedSomething ))
            (not (= ?doorRegion ?otherDoorRegion ))
            (affords-removing-from-switch ?switchRegion )
            ; non-releaser implies that passage cannot be perceived
            (not (passable ?doorRegion ?otherDoorRegion ))
            (not (passable ?otherDoorRegion ?doorRegion )))
    :effect
```

```
        (and
            ; the switch will afford to be triggered
            (switch-triggerable ?switchRegion )
            ; it will not afford to remove something from it anymore
            (not (affords-removing-from-switch ?switchRegion ))
            ; Door was not open, item was no releaser
            (hasLiftedSomething )))


(:action change-room
    :parameters (?doorRegion - doorRegion
                 ?targetDoorRegion - doorRegion )
    :precondition
        (and
            (robotAt ?doorRegion)
            (not (= ?doorRegion ?targetDoorRegion ))
            (passable ?doorRegion ?targetDoorRegion ))
    :effect
        (and
            (not (robotAt ?doorRegion ))
            (robotAt ?targetDoorRegion ))))
```

B problem.pddl

```
(define (problem macs-prob)

        (:domain macs-example)

        (:objects
           region1_left region2_left region1_right - region
           switchRegion - switchRegion
           doorRegionLeft doorRegionRight - doorRegion
           rightRoom - room
           leftRoom - switchRoom )

        (:init
           (robotAt region1_left)
           (inRoom region1_left leftRoom )
           (inRoom region2_left leftRoom )
           (inRoom switchRegion leftRoom )
           (inRoom doorRegionLeft leftRoom )
           (inRoom region1_right rightRoom )
           (inRoom doorRegionRight rightRoom )

           (switch-releaser-liftable region2_left )
           (non-releaser-liftable region1_right )
           (affords-removing-from-switch switchRegion ))

        (:goal
            (and (non-releaser-liftable region1_left )
                 (non-releaser-liftable switchRegion )
                 (switch-triggerable switchRegion )
                 (not (hasLiftedSomething )))))
```