Planning

Aim The aim of this lab is for you to investigate how planning works, and to understand how to formalize the world where the agent is situated in a logical way.

Preparation Go through the slide material of Lecture 8 and 9 about planning on the course website. Try out also the STRIPS planner with the Blocks World example.

About the lab This lab consists of three parts, and it can be done by at most two persons.

Lab examination demonstrate the tasks you solved to Pierangelo during any scheduled lab session.

Grading Task A → 3 / Tasks A, B → 4 / Tasks A, B, C → 5

Task A - Shakey the Robot

Shakey the robot (1966-1972 SRI International) was the first general-purpose mobile robot to be able to reason about its own actions. While other robots would have to be instructed on each individual step of completing a larger task, Shakey could analyze the command and break it down into basic chunks by itself.

The robot's programming was primarily done in LISP. The STRIPS planner it used was conceived as the main planning component for the software it utilized. As the first robot that was a logical, goal-based agent, Shakey experienced a limited world. A version of Shakey's world could contain a number of rooms connected by corridors, with doors and light switches available for the robot to interact with. Shakey had a short list of available actions within its planner. These actions involved traveling from one location to another, turning the light switches on and off, opening and closing the doors, climbing up and down from rigid objects, and pushing movable objects around.[1]

The original STRIPS program was designed to control Shakey the robot. Figure 2 shows a version of Shakeys world consisting of four rooms lined up along a corridor, where each room has a door and a light switch. The actions in Shakeys world include moving from place to place, pushing movable objects (such as boxes), climbing onto and down from rigid objects (such as boxes), and turning light switches on and off. The robot itself was never dexterous enough to climb on a box or toggle a switch, but the STRIPS planner was capable of finding and printing out plans that were beyond the robots abilities. Shakeys actions are listed in Figure 1.

[1] http://en.wikipedia.org/wiki/Shakey_the_robot

- **go(x, y)**, which requires that Shakey be at x, and that x and y are locations in the same room. By convention a door between two rooms is in both of them.
- Push a box b from location x to location y within the same room: **push(b, x, y)**. You will need the predicate box(...) and constants for the boxes.
- For Shakey to be able to find a box in a room, the light must be on in that room.
- To switch the light in a room on or off, Shakey must climb onto a box that is positioned below the switch (otherwise he can not reach the switch).
- Climb onto a box **climbUp(b)**; climb down from a box **climbDown(b)**. You will need the predicate on and the constant floor.
- Turn a light switch on **turnOn(s)**; turn it off **turnOff(s)**. To turn a light on or off, Shakey must be on top of a box at the light switchs location.

Figure 1: Shakey's actions

In this task do the following[2]:

- Download the STRIPS planner from the course website.
- Describe Shakeys actions and the world from Figure 2 in STRIPS notation.
- Run the planner with your description of the world to find a plan:
  1. to move Shakey from room3 to room1.
  2. to switch off the light in room1
  3. to get box2 into room2.

Task B - Planning with Constraints

To increase the flexibility in planning, in this task we investigate the use of constraints. The idea is to avoid unnecessary commitments (as done in linear planning) as much as possible (least commitment policy).

We consider the Colored Blocksworld. This domain consists of a set of objects of two types; blocks and pyramids. Objects are colored; orange, green or blue. Assume we have four blocks and two pyramids sitting on a table, and colored as illustrated in Fig.3. Objects can be stacked, but only on top of a block (that is, no object can be stacked on top of a pyramid). A robot arm can pick up an object and move it to another position, either on the table or on top of a block. The arm can pick up only one object at a time.

The goal of this task is to make a plan to build the stack of colored objects given in Fig. 4. Note that the goal state is general since it requires to build a stack of three objects without saying which ones, only the color. To solve this task you can use the IDstrips.pl planner written in SWI-Prolog that supports constraints.

[2]Note that the STRIPS planner we use is a simple progression planner, and it is not that efficient. This task does not require you to find the optimal plan.

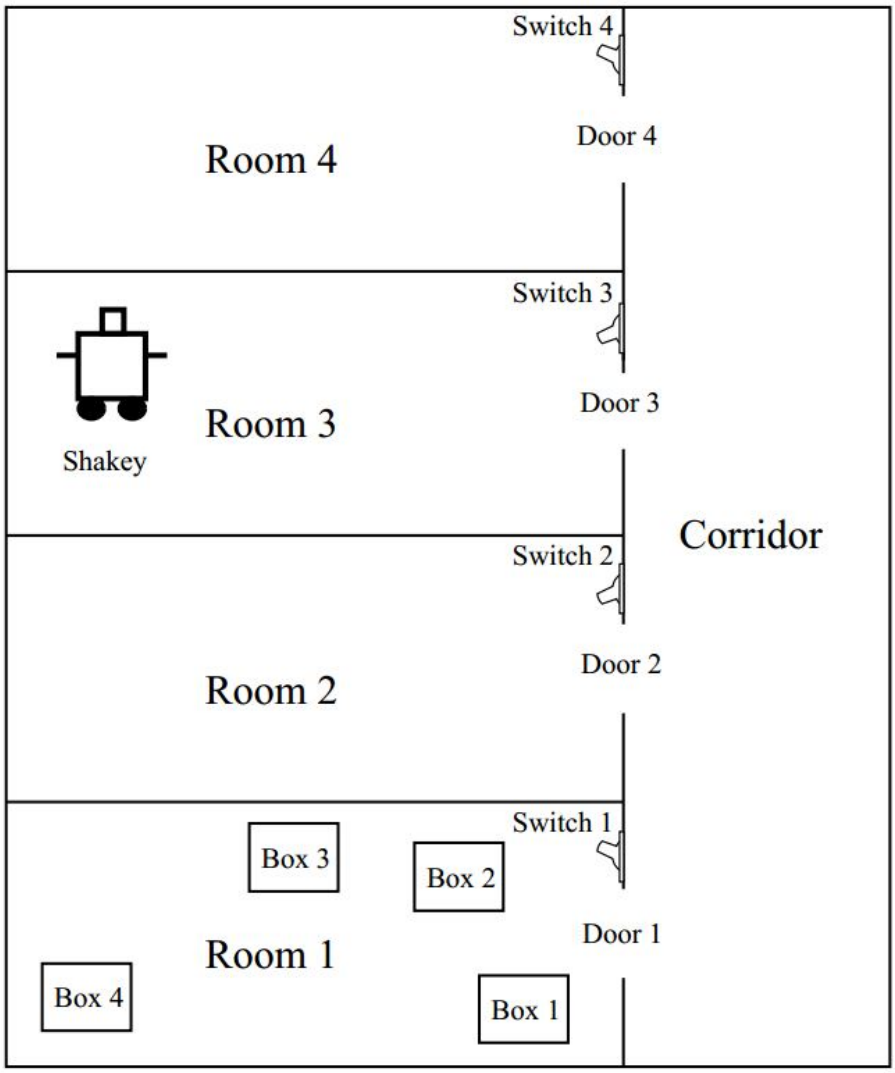

Figure 2: Shakeys world. Shakey can move between landmarks within a room, can pass through the door between rooms, can climb climbable objects and push pushable objects, and can flip light switches.

- To be able to use constraints in SWI-Prolog, write at the beginning of your working file: `:- use_module(library(clpfd)).`
- The library clpfd works on integers not letters; so use integers to name the objects.
- Add the necessary constraints to the solver at the beginning before doing any planning. Doing so will improve the efficiency of the planner.
- Use constraints for equality (`#=`), disequality (`#\=`), greater (`#>`), etc.
- Model block, pyramid, and colors as constraints. For example, to declare that the objects numbered 1 to 4 are blocks, do:

  ```
  block(X) :- X in 1..4.
  ```

- To declare that the objects 1, 2 and 5 are green, do:

  ```
  green(X) :- X in 1..2 \/ 5.
  ```

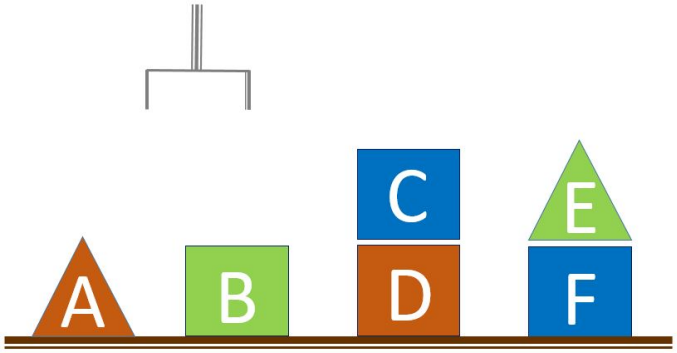

Figure 3: Colored Blocksworld; initial state

Goal State	**on(X,Y)**
	on(Y,Z)
	green(Y)
	blue(Z)

Figure 4: Colored Blocksworld; final state

- Write actions as:

```
act(move(X,A,B),[...],[...],[...]):-
block(X),
A #\= B.
```

- Write goal states as:

```
goal_state([on(X,Y)]):-
green(X).
```

Task C - Partial Order Planning

Partial order planning is an approach to planning that leaves decisions about the ordering of actions as open as possible (Principle of Least Commitment). It contrasts with total order planning, which produces an exact ordering of actions. Given a planning problem, a partial order plan specifies all actions that need to be taken, but specifies an ordering of the actions only where necessary. A partial order planner is a planner which constructs a partial order plan and search for a solution. The input is the problem description, consisting of descriptions of the initial state, the goal and possible actions.

In this Task we will work with the partial order planner pop.pl that you can download from the course website. Figure 5 shows the preparatory work for this task.

- Download pop.pl and tv.pl from the course website.
- Run SWI-prolog and consult them:

  `?- [pop,tv].`
- Open the file tv.pl with any text editor and look at the problem description. Note that the action definitions do not have any precondition and have an empty delete list.
- Query the planner to find the partial order plan for the goal:

  `?- solve([sated,readyForExam,watchedTV],4).`
- Use only the fluents defined in tv.pl to fix the actions definitions; precondition and delete list.
- Consult your modified version of tv.pl and run the planner to derive all the partial plans for the query above.

Figure 5: Preparatory work for Task C

In this task do the following:

1. Download pop.pl and shopping.pl from the course website.
2. Run SWI-prolog and consult them:

   `?- [pop,shopping].`
3. Query the planner to find the total order plans that satisfy the goals (a-c):

```
(a)  ?- solve([has(chris,drill)],2).
(b)  ?- solve([has(chris,drill),has(chris,banana)],4).
(c)  ?- solve([has(chris,bread),has(chris,cheese)],3).
```

4. Query again the goals (a-c) at Step 3 but with the aim to find a partial order plan. Consider the case where everything can be run in parallel.
5. Are the plans found at Step 4 "conceptually" correct? If not, can you correct them?
6. Extend shopping.pl to include a new action `carry(A,W,X,Y)` meaning that the agent A carries an item W from the location X to the location Y.

   Introduce the fluent `objAt(W,X)` meaning that the object W is at location X.

   Note: if an agent A has an item W and goes from X to Y, then W is not carried on with the agent. To do so, you need to execute the action carry.
7. Consult your modified version of shopping.pl and run the planner to find the partial order plans to the goals:

```
(d)  ?- solve([objAt(bread,home)],3).
(e)  ?- solve([objAt(bread,home),objAt(cheese,home)],6).
```