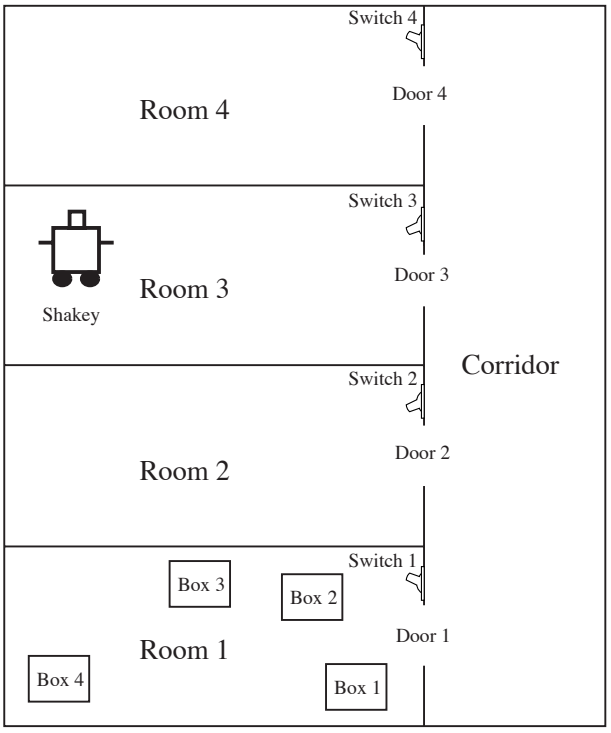
Agent-Based Systems
Tutorial 3

Suggested solutions

Michael Rovatsos

Suggestions for solutions and hints are printed in italics below each question

Q1 (Adapted from Russell & Norvig) Shakey the robot was the application for which the STRIPS language was originally developed. The figure below shows a version of Shakey's world consisting of four rooms and a corridor. Each room has a door and light switch. Shakey can move from place to place, push movable objects, climb on (and down from) rigid objects and turn light switches on and off (actually the real Shakey couldn't, but the planner can handle all these actions). Shakey needs to climb on a box to turn a light switch, and we assume that all rooms are connected by doors which belong to both rooms.



1. Develop a logical language for describing Shakey's world
2. Describe Shakey's actions by appropriate action schemata

3. Formally describe the initial state shown in the figure above
4. Construct a plan for Shakey to get Box_2 into $Room_2$ from this initial state

Solution suggestions: *Obviously, many different designs are possible here. The one described here is inspired Russell & Norvigs's exercise (p. 414 in the R&N textbook, 2nd ed.)*

1. *We use the following predicates to describe the domain:*
   - $In(o, x)$ *to describe that Shakey/box/switch is in location* x*,*
   - $Light(x)$ *to denote that the light is on in location* x*,*
   - $On(o, p)$ *to denote that object (or Shakey)* o *is stacked on object, and*
   - $Box(o)$ *to denote the object* o *is a box.*

   As constants for locations we have $Room_1, \ldots, Room_4, Corridor$ *for locations,* $Shakey$ *and* $Box_1, \ldots, Box_4$ *for objects,* $Switch_1, \ldots, Switch_4$ *for switches, and* $Floor$ *as a special constant for the floor of the building.*
2. *Shakey's six actions can be described as follows:*

$\mathbf{Go}(x, y)$
$pre\ \{In(Shakey, x), Location(y)\}$
$del\ \{In(Shakey, x)\}$
$add\ \{In(Shakey, y))\}$

$\mathbf{Push}(b, x, y)$
$pre\ \{Box(b), In(b, x), In(Shakey, x), Location(y)\}$
$del\ \{In(b, x), In(Shakey, x)\}$
$add\ \{In(b, y), In(Shakey, y)\}$

$\mathbf{ClimbUp}(b)$
$pre\ \{Box(b), In(b, x), In(Shakey, x), On(Shakey, Floor)\}$
$del\ \{On(Shakey, Floor)\}$
$add\ \{On(Shakey, b)\}$

$\mathbf{ClimbDown}(b)$
$pre\ \{Box(b), On(Shakey, b)\}$
$del\ \{On(Shakey, b)\}$
$add\ \{On(Shakey, Floor)\}$

$\mathbf{TurnOn}(s)$
$pre\ \{On(Shakey, b), In(b, x), Switch(s, x), \neg Light(x)\}$
$del\ \{\}$
$add\ \{Light(x)\}$

$\mathbf{TurnOff}(s)$
$pre\ \{On(Shakey, b), In(b, x), Switch(s, x), Light(x)\}$
$del\ \{Light(x)\}$
$add\ \{\}$

Notes:

- *The* $Location$ *predicate is introduced to avoid attempts to move to a box or a switch which would be nonsensical. The same is true for the* Box *and* $Switch$ *predicates. You can point out that generally variables can be bound to any constant in the universe. Students with a knowledge of logic may worry about decidability problems with computing preconditions. You can point out that these can be avoided here since variables only range over finite sets of objects in STRIPS (in fact, there are even more restrictions in the language, but this one would be sufficient to make the language essentially equivalent to propositional (rather than first-order) logic).*

- *The model above is simplified because it assumes the robot can climb on something or push something as soon as it is in the same room. A more fine-grained model might use, two-dimensional coordinates for example. You can discuss what this would entail, e.g. making decisions about the granularity of the grid that is used, defining demarcation lines between the rooms and using two levels of abstraction for actions (I would still like to be able to say "go to room 3").*
- *You may want to discuss the advantages and disadvantages of having actions that are inverse to each other as in the above example. The main advantage is that the agent never "gets stuck" (it can always undo its actions), but this also means that if simple search is used, every undo action will be equally considered in each step including undoing what one has just achieved (but there are techniques around this, e.g. goal decomposition (trying to achieve the literals of the goal state one by one incrementally)).*
- *A final thing to note is that this simple kind of planning does not consider uncertainty (the world is deterministic), incomplete knowledge (we have to know the complete initial state to derive a plan, unlike, e.g. in the vacuum world example), hierarchical decomposition (all actions are "primitive" actions, they cannot be grouped together to refine plans depending on the level of granularity required in a given situation – e.g. if we have a plan for building a house, we are not going to plan each step in detail until obtaining planning permission i.e. achieving a first significant sub-goal), re-planning and execution monitoring (no provisions are made for actions that may fail or that the world state may change while the plan is being executed).*

3. *Initial state:*

$$\{In(Shakey, Room_3), On(Shakey, Floor), Light(Room_4), Light(Room_1),\\ In(Box_1, Room_1), \ldots, In(Box_4, Room_1), Box(Box_1), \ldots, Box(Box_4),\\ Switch(Switch_1, Room_1), \ldots, Switch(Switch_4, Room_4),\\ Location(Room_1), \ldots, Location(Room_4)\}$$

4. *Possible plan:*

$$\{Go(Room_3, Room_1), Push(Box_2, Room_3, Room_2)\}$$

Discuss notions of acceptability and correctness of a plan using this example. For this purpose, it may be useful to trace the iterative modifications to the knowledge base describing the initial state until the goal state is reached. It should also be explained that the goal state obviously only needs to be a subset of the knowledge base in some future state.

Q2 The Advanced Thermostat System: Assume a heater control system controls a living room and a bedroom area, and is equipped with

- separate thermostats and heater units for both rooms
- timer clocks that allow the thermostat settings to be overruled at certain times (by binary "on"/"off" settings) and also time to be sensed
- motions sensors to note whether people are present in the room

Your task is to build an intelligent BDI-based agent using the control loop discussed in the lecture:

Practical Reasoning Agent Control Loop
1. $B \leftarrow B_0$; $I \leftarrow I_0$; /* initialisation */
2. while $true$ do
3. get next percept ρ through $see(\ldots)$ function
4. $B \leftarrow brf(B, \rho)$; $D \leftarrow options(B, I)$; $I \leftarrow filter(B, D, I)$;
5. $\pi \leftarrow plan(B, I, Ac)$;
6. while not ($empty(\pi)$ or $succeeded(I, B)$ or $impossible(I, B)$) do
7. $\alpha \leftarrow head(\pi)$;
8. $execute(\alpha)$;
9. $\pi \leftarrow tail(\pi)$;
10. get next percept ρ though $see(\ldots)$ function
11. $B \leftarrow brf(B, \rho)$;
12. if $reconsider(I, B)$ then
13. $D \leftarrow options(B, I)$; $I \leftarrow filter(B, D, I)$
14. if not $sound(\pi, I, B)$ then
15. $\pi \leftarrow plan(B, I, Ac)$
16. end-while
17. end-while

For complex actions you can assume a plan library that will return a sequence of actions to achieve any achievable goal from any given initial state.

1. Design a language for describing the advanced thermostat world formally in terms of percepts, actions, and beliefs
2. Suggest a suitable agent design by formally defining the brf, $options$, $filter$ and $reconsider$ functions
3. For a sequence of percepts of your choice of a few steps, sketch the operation of your system by tracing the workings of the BDI control loop using the functions you have designed

Solution suggestions:

1. *In the following definitions we use B as a subscript for bedroom and L for the living room. The percept set Per is going to consist of tuples of the form*

$$[time, t_l, t_b, m_l, m_b, action_l, action_b]$$

where $time \in Time$ is the current time (e.g. in hours and minutes $hh : mm$ format), t_i is the temperature (in degrees centigrade) measured in each room ($i \in \{b, l\}$), $m_i \in \{true, false\}$ is the boolean reading of the motion sensor in each room, and $action_i \in \{on, timer, off, settemp(Temp), add(Start, End), remove(Start, End)\}$ is the action currently taken by the user for the heating controller of every room. on and off turn the heating on and off, and overrule any existing timer events until the setting is set back to $timer$. $settemp$ sets the (user-)desired temperature in the respective room to $Temp$, add and $remove$ add and remove a timer setting from time $Start \in Time$ to time $End \in Time$ at the respective controller.
For the belief set, we are going to use tuples of the form

$$[state_l, state_b, presence, settemp_l, settemp_b, timerstore_l, timerstore_b]$$

where $state_i \in hot, ok, cold, on, off, timer$ *denotes whether the temperature is OK in the respective room (*$i \in \{l, b\}$*) or it is too coold/too hot, and also whether the timer has been overruled by the heating being set to* on *or* off *by the user, or* $timer$ *in case a timer event is currently occurring (note difference to "timer" action by user).* $presence$ *is a boolean value denoting that someone is in the house,* $settemp_i$ *is an internal store for the target temperature set by the user in the room, and* $timerstore_i = [(s_1, e_1), \ldots, (s_n, e_n)]$ *is a store for the timer events set for the respective controller by the user. (Note that our definition allows use of a singleton belief set since different beliefs are mutually exclusive.*

2. *In these definitions we are going to use a schematic representation using wildcards * wherever some values in the tuples are not affected. We also use functions* $add(L, i)$ *and* $remove(L, i)$ *to denote addition/removal of an element to/from a list. We first define the belief revision function as*

$$\begin{aligned} &brf([state_l, state_b, presence, settemp_l, settemp_b, timerstore_l, timerstore_b], \\ &\quad [time, t_l, t_b, m_l, m_b, action_l, action_b]) = \\ &\quad [state'_l, state'_b, presence', settemp'_l, settemp'_b, timerstore'_l, timerstore'_b] \end{aligned}$$

where

- $state'_i = hot$ *if* $t_i > settemp_i$ *and* $state'_i = cold$ *if* $t_i < settemp_i$*,* $state_i = ok$ *if* $t_i = settemp_i$*, and* $state_i \in \{hot, ok, cold\}$ *or* $action_i = timer$ *holds (*$\{hot, cold, ok\}$ *are only possible if the timer is not overruled or the controller was in one of those states previously),*
- $state'_i = timer$ *if* $\exists (S, E) \in timerstore_i . S \prec Time \prec E$ *(where* $\prec$ *means "before" in a temporal sense) and* $state_i \in \{hot, ok, cold, timer\}$*,*
- $state'_i = on$ *if* $action_i = on$*,* $state'_i = off$ *if* $action_i = off$*,*
- $presence' = true$ *if* $m_l \vee m_b$*,*
- $settemp'_i = Temp$ *if* $action_i = settemp(Temp)$*,*
- $timerstore'_i = add(timerstore_i, (S, E))$ *if* $action_i = add(S, E)$*,*
- $timerstore'_i = remove(timerstore_i, (S, E))$ *if* $action_i = remove(S, E)$

(elements of the belief set not mentioned remain unchanged).
For desires, we are going to use a simple set

$$D = \{MoneySaved, RightTemp, UserWishFulfilled\}$$

and for intentions

$$I = \{SaveGas, RightTempReached, TimerEventsFulfilled, CommandsFulfilled\}$$

. With this, the $options$ *function can be defined as follows:*

$$options([state_l, state_b, presence, settemp_l, settemp_b, timerstore_l, timerstore_b], I) = D$$

where

- $D = \{UserWishFulfilled\}$ *if* $\exists i . state_i \in timer, on, off$ *(there should only be one option if we have reached a set heating time or the user has turned any controller on or off),*

– *Else: if* $presence = false$*,* $D = \{MoneySaved\}$*; if* $presence = true$*,* $D = \{RightTemp\}$

Note that in this simple example, the options generated don't depend on current intentions I *since essentially these can change in every instance, i.e. no desires are ruled out because previous the agent has committed itself to previous intentions.*
Next, for the filter function, letting

$$B = \{[state_l, state_b, presence, settemp_l, settemp_b, timerstore_l, timerstore_b]\}$$

we can define:

– $filter(B, \{RightTemp\}, \{RightTempReached\}) = \{RightTempReached\}$ *if* $state_i \in \{cold, hot\}$ *for any* $i \in \{l, b\}$ *and* $presence = true$*;*
$filter(B, \{RightTemp\}, \{RightTempReached\}) = \{SaveGas\}$ *else (note that this does not distinguish between different rooms, and might cause sub-optimal behaviour),*
– $filter(B, \{MoneySaved\}, \{RightTempReached\}) = \{SaveGas\}$*,*
– $filter(B, \{UserWishFulfilled\}, I) = \{TimerEventsFulfilled, CommandsFulfilled\}$ *if* $I \notin \{TimerEventsFulfilled, CommandsFulfilled\}$
– ...

A simple proposal for the $reconsider$ *function is to set it to true if any of the following conditions occur:*

– $presence$ *becomes* $true$ *when it was* $false$ *before and vice versa (can we actually express that with the language used? – yes, we can implicitly, because the "reconsider" function also depends on previous intentions, and if the previous intention was* $SaveGas$ *we know that nobody is in);*
– *a timer event is modified while it is being executed,*
– *the set temperature is modified by the user during plan execution (can this be identified from the way in which we express beliefs and percepts?).*

The proposed solution is certainly not an ideal solution and it's not even complete. Discuss its shortcomings and related design issues, including what kind of planning library would have to be provided for this domain. The primary goal of this exercise is for people to get a feel for BDI-style agent design and to understand that it can be tedious process even for such simple systems.