CS322 Fall 1999
Module 9 (Representing Actions)
Assignment 9

Solution.

Question 1

Consider the domain of assignment 2, as shown in Figure 1.

Suppose, at each time step, one of the following actions can be carried out: turn on a specific tap, turn off a specific tap, put a plug in the sink, put a plug in the bath, take the plug out of the sink, take the plug out of the bath, or wait.

Suppose initially all taps are off, and the bath and sink are unplugged, and $p1$ is pressurized. $p1$ remains pressurized for all actions.

(a) Axiomatize, using the situation calculus, how *on*, *plugged* and *unplugged* are affected by the actions.

(b) Axiomatize the domain so that you can determine whether there is flow in the drains for various situations. You need to consider both the bath and the sink, but you don't need to consider how the floor gets wet.

(c) Give axioms that let you derive *noflow* for the shower, into the bath and in the drains. *noflow* is true when *flow* is not true. You may **not** use negation as failure. You can define other predicates if you like.

(d) Suppose the plugged bath with water flowing in goes from empty to half full in one time step (i.e., in the time it takes to execute one action), and from half-full to full in one time step, and will overflow wetting the floor in the next time step. The unplugged bath empties on one time step. The plugged sink fills up in one time step and, in the next time step it overflows with water flowing in. The unplugged sink empties on one time step. Intuitively, think of the action as doing the command, then waiting a bit (enough time for the sink to fill or the bath to half fill).

The floor, once wet, remains wet.

Axiomatize how the floor can be wet in different situations. You can define whatever predicates you may need.

You should try your axiomatization in CILog. The only built-in predicate you may use is inequality (`\=`).

Solution to part (a)

Axiomatize, using the situation calculus, how *on*, *plugged* and *unplugged* are affected by the actions.

These are all primitive, and so we axiomatize how they are true initially and how they are true after an action depending on what's true before the action.

```
on(T,do(turnon(T),S)) <- poss(turnon(T),S).
on(T,do(A,S)) <- poss(A,S) & on(T,S) & A \= turnoff(T).

unplugged(B,do(unplug(B),S)) <- poss(unplug(B),S).
```

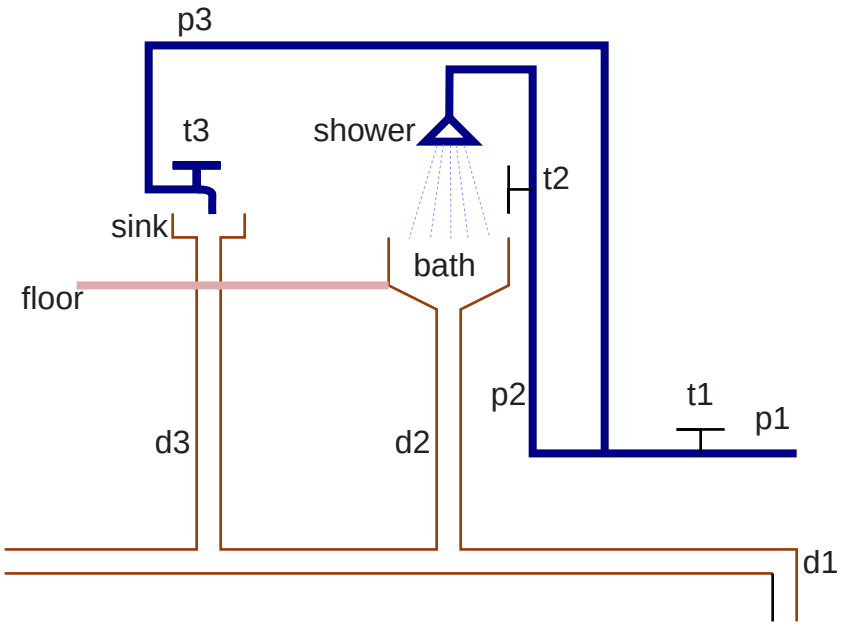

Figure 1: The Plumbing Domain

```
unplugged(B,do(A,S)) <-  poss(A,S) & unplugged(B,S) & A \= plug(B).
unplugged(B,init).

plugged(B,do(plug(B),S)) <- poss(unplug(B),S).
plugged(B,do(A,S)) <- poss(A,S) & plugged(B,S) &  A \= unplug(B).
```

We also need to state that all actions are always possible:

```
poss(wait,S).
poss(turnon(T),S).
poss(turnoff(T),S).
poss(unplug(B),S).
poss(plug(B),S).
```

As all of the actions are always possible, one could omit all mention of *poss*, but then the representation doesn't let us easily add another action that does have a prerequisite.

Alternatively, you could give reasonable preconditions:

```
poss(wait,S).
poss(turnon(T),S) <- off(T,S).
poss(turnoff(T),S) <- on(T,S).
poss(unplug(B),S) <- plugged(B,S).
poss(plug(B),S) <- unplugged(B,S).
```

Solution to part (b)

Axiomatize the domain so that you can determine whether there is flow in the drains for various situations. You need to consider both the bath and the sink, but you don't need to consider how the floor gets wet.

We can just copy the solution to assignment 2, adding a situation term to each predicate. Note that all of these can be considered to be derived predicates. [I also added a flow from the sink spout.]

```
pressurised(p1,init).
pressurised(p1,do(A,S)) <- pressurised(p1,S).
pressurised(p2,S) <- on(t1,S) & pressurised(p1,S).
pressurised(p3,S) <-  on(t1,S) & pressurised(p1,S).
flow(shower,S) <- on(t2,S) & pressurised(p2,S).
wet(bath,S) <- flow(shower,S).
flow(sink,S) <- on(t3,S) & pressurised(p3,S).
wet(sink,S) <- flow(sink,S).
flow(d2,S) <- wet(bath,S) & unplugged(bath,S).
flow(d1,S) <- flow(d2,S).
flow(d3,S) <- wet(sink,S) & unplugged(sink,S).
flow(d1,S) <- flow(d3,S).
```

Solution to part (c)

Give axioms that let you derive *noflow* for the shower, into the bath and in the drains. *noflow* is true when *flow* is not true. You may **not** use negation as failure. You can define other predicates if you like.

We need to know when a tap is off, and this must be primitive:

```
off(t3,init).
off(t2,init).
off(t1,init).
off(T,do(turnoff(T),S)) <-
   poss(turnoff(T),S).
off(T,do(A,S)) <-
   poss(A,S) &
   off(T,S) &
   A \= turnon(T).
```

We can then axiomatize how flows work:

```
noflow(sink,S) <-
    off(t3,S).
noflow(sink,S) <-
   unpressurised(p3,S).
unpressurised(p3,S) <-
   off(t1,S).
unpressurised(p3,S) <-
   unpressurised(p1,S).
noflow(shower,S) <-
    off(t2,S).
noflow(shower,S) <-
   unpressurised(p2,S).
unpressurised(p2,S) <-
   off(t1,S).
unpressurised(p2,S) <-
   unpressurised(p1,S).
```

Solution to part (d)

Suppose the plugged bath with water flowing in goes from empty to half full in one time step (i.e., in the time it takes to execute one action), and from half-full to full in one time step, and will overflow wetting the floor in the next time step. The unplugged bath empties on one time step. The plugged sink fills up in one time step and, in the next time step it overflows with water flowing in. The unplugged sink empties on one time step. Intuitively, think of the action as doing the command, then waiting a bit (enough time for the sink to fill or the bath to half fill).

The floor, once wet, remains wet.

Axiomatize how the floor can be wet in different situations. You can define whatever predicates you may need.

The tricky thing here is thinking about how the world works; what has to be true before and after the action. For example, the way the problem is stated, the bath fills if it is plugged and water is flowing in.

```
empty(bath,S) <-
   unplugged(bath,S).
empty(bath,do(A,S)) <-
   poss(A,S) &
   empty(sink,S) &
   noflow(shower,do(A,S)).

halffull(bath,do(A,S)) <-
   poss(A,S) &
   halffull(bath,S) &
   plugged(bath,do(A,S)) &
   noflow(shower,do(A,S)).
halffull(bath,do(A,S)) <-
   poss(A,S) &
   plugged(bath,do(A,S)) &
   empty(bath,S) &
   flow(shower,do(A,S)).

full(bath,do(A,S)) <-
   poss(A,S) &
   plugged(bath,do(A,S)) &
   full(bath,S).
full(bath,do(A,S)) <-
   poss(A,S) &
   plugged(bath,do(A,S)) &
   halffull(bath,S) &
   flow(shower,do(A,S)).

overflow(bath,do(A,S)) <-
   poss(A,S) &
   flow(shower,do(A,S)) &
   full(bath,S).

empty(sink,S) <-
   unplugged(sink,S).
```

```
empty(sink,do(A,S)) <-
   poss(A,S) &
   empty(sink,S) &
   noflow(sink,do(A,S)).

full(sink,do(A,S)) <-
   poss(A,S) &
   plugged(sink,do(A,S)) &
   empty(sink,S) &
   flow(sink,do(A,S)).
full(sink,do(A,S)) <-
   poss(A,S) &
   plugged(sink,do(A,S)) &
   full(sink,S).

overflow(sink,do(A,S)) <-
   poss(A,S) &
   flow(sink,do(A,S)) &
   full(sink,S).

wet(floor,S) <-
   overflow(bath,S).
wet(floor,S) <-
   overflow(sink,S).
wet(floor,do(A,S)) <-
   wet(floor,S) &
   poss(A,S).
```

Question 2

(a) Suppose you considered using the STRIPS representation for this domain. Which predicates would be static, which would be primitive and which would be derived?

(b) Give the STRIPS representation for the action *turnon*($t3$).

(c) Give the STRIPS representation for the action *wait*.

(d) Is STRIPS or the situation calculus more natural for representing this domain? Explain why.

Solution

(a) Suppose you considered using the STRIPS representation for this domain. Which predicates would be static, which would be primitive and which would be derived?

No predicates are static.

The derived predicates are: *pressurised*, *flow*, *noflow*, *unpressurised*.

The primitive predicates are: *on*, *off*, *plugged*, *unplugged*, *empty*, *halffull*, *full*, *overflow* and *wet*.

(b) Give the STRIPS representation for the action *turnon*($t3$).

The obvious solution is:

```
preconditions: off(t3)     (or perhaps true)
addlist: [on(t3)]
deletelist: [off(t3)]      (or perhaps []).
```

However, this isn't quite right, as it doesn't fully specify its effect on *empty*, *halffull*, *full*, etc. There is no simple representation in STRIPs, as whether the floors gets wet depends on other conditions.

We would like something like a conditional addlist: if sink is full and t1 is on then add *wet*(*floor*).

(c) Give the STRIPS representation for the action *wait*.

Again, the obvious solution is:

```
preconditions: true
addlist: []
deletelist: []
```

But, if there is water flowing in the bath, and the bath is half full, then *full*(*bath*) should be added and *halffull*(*bath*) should be removed, and *full*(*bath*) were true then add *wet*(*floor*), etc.

(d) Is STRIPS or the situation calculus more natural for representing this domain? Explain why.

I would say that the situation calculus is more natural as all of the rules are local. The problem with STRIPs is that the effect of actions can propagate to be non local. For example, one effect of *turnon*($t1$) could be to wet the floor (if $t2$ were on and the bath is full). In the situation calculus this is taken care of, but for STRIPs we need to say directly how it affects each primitive (and *wet*(*floor*) needs to be primitive).