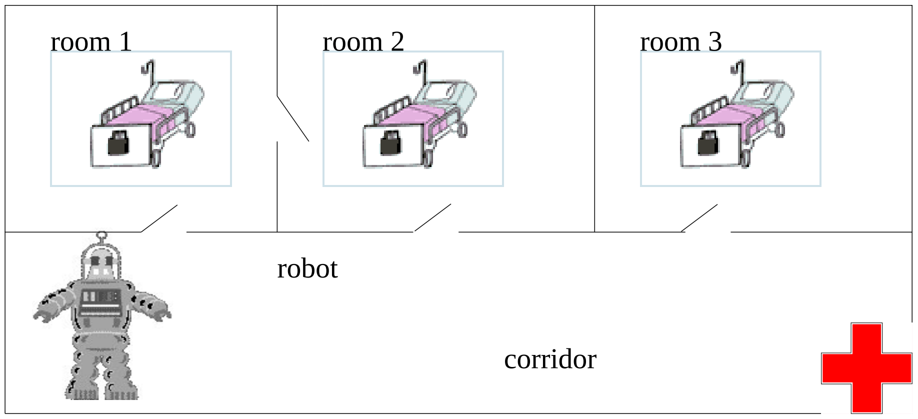
Comp 3620: Artificial Intelligence
Lab 3: Situation Calculus

March 31, 2005

1 Introduction

The goal of this lab is to create a very basic planning system using the situation calculus (see Lecture KRR6). This planner will be implemented using PROLOG (see Lecture KRR5). The purpose of this planner is to generate plans for a medical assistant robot. **Firstly, read completely the document**. The result of this lab should be the submission of three PROLOG programs with comments: `robot1.pl`, `robot2.pl`, `robot3.pl`.



The robot is in charge of three rooms. In each room, there is a patient. The patient can be either hungry or thirsty or both. The robot is initially in the corridor. The robot can move from one location to another location if there is a door between the two locations: for instance, the robot can move from the corridor to room 1 and from room 1 to room 2, but it cannot go from room 2 to room 3, it has to go back to the corridor first. The robot can bring food to the patient if the patient is hungry and it can bring water if the patient is thirsty. After bringing the food, the patient is not hungry anymore. After bringing water, the patient is not thirsty anymore.

2 Preliminaries

In order to become more familiar with SWI-prolog, you can first have a look to the file `goodteacher.pl`.

- Run `swipl`
- You see the prolog prompt: to load the file enter the following fact

```
[goodteacher].
```

- Then you can ask queries: try `goodteacher(yannick).` then try `goodteacher(T).`.

- PROLOG is based on a backward chaining algorithm. You can *trace* the way PROLOG tries to answer the queries. Try `trace.` then `goodteacher(T).`.
- To quit PROLOG, the command is `halt.`

3 Your Work

3.1 Knowledge representation 1 (easy)

For this first part, we do not consider that the patients are thirsty or hungry and we consider that the robot can only move from locations to locations.

Step 1: Using the Situation Calculus (situations, facts, fluents, axioms...), write in PROLOG the facts and the predicates that represent the world described in the first section. In your implementation, how does a situation term look like?

In order to test if your implementation works, you can write then a set of predicates like

```
goal(S) :- fluent1(...,S),....,fluentN(...,S).
```

Such a predicate says that the situation S is a goal if `fluent1(...,S)...  fluentN(...,S)` are true.

Example: if we want the robot in room 1, then the predicate `goal(S)` is something like:

`goal(S) :- pred(...S)` where `pred` is a fluent which says that in the situation `S` the robot is in room 1.

Given this goal, here are some queries `goal(..)` where `..` is a term which represents the following situations:

1. The robot was initially in the corridor then it moves to room 1.
2. The robot was initially in the corridor then it moves to room 2.
3. The robot was initially in the corridor then it moves to room 2, then it moves to room 1.
4. The robot was initially in the corridor then it moves to room 3, then it moves to room 1.

Step 2: For each situation, write the query in PROLOG. The result should be either `Yes` or `No` depending on the situation. Write this result in your PROLOG file (as a comment) with a small explanation about the result.

Step 3: Is there any frame problem here? Explain.

Submission: the result of this section will be a file `robot1.pl` with your representation in PROLOG and the answers to the questions as comments.

3.2 Knowledge representation 2 (medium)

For this second section, we consider the complete world. First of all, copy `robot1.pl` in a new file `robot2.pl`. For each step, TEST YOUR PROGRAM by making several queries.

Step 4: Write the fact that patients can be hungry. Write the axioms about the action “bring food” of the robot. Is there any frame problem?

Step 5: We consider the following initial situation: "The robot is in the corridor and only the patient of room 2 is hungry". We consider the goal "The patient of room 2 is not hungry". Write the initial situation and ask the queries to PROLOG with the following situations:

1. The robot moves to room 2 and brings food to the patient of room 2.
2. The robot moves to room 1 then to room 2 and brings food to patient of room 2.
3. The robot moves to room 3 and brings food to patient of room 3.
4. The robot moves to room 3, brings food to patient of room 3, moves to the corridor, moves to room 2, brings food to patient 2.

For each situation, ask the query to PROLOG and put a comment in the file `robot2.pl` about the result and your explanation (or expectation) about the result.

Step 6: Write the fact that patients can be thirsty. Write the axioms about the action "bring water" of the robot.

Step 7: We consider the following initial situation: "The robot is in the corridor and the patients of room 2,3 are hungry, the patients of room 1,2 are thirsty". We consider the goal "The patient of room 1 is not thirsty and the patient of room 2 is neither thirsty nor hungry". Write the initial situation and ask the queries to PROLOG with the following situations:

1. The robot moves to room 1, brings water to the patient of room 1, moves to room 2 and brings food to the patient of room 2, brings water to the patient of room 2.
2. The robot moves to room 1 then to room 2 and brings food to patient of room 2, moves to room 1, brings water to patient of room 1, moves to room 2, brings water to patient of room 2.
3. The robot moves to room 3 and brings food to patient of room 3, moves to corridor, moves to room 1, brings water to the patient of room 1, moves to room 2 and brings food to the patient of room 2, brings water to the patient of room 2.

For each situation, ask the query to PROLOG and put a comment in the file `robot2.pl` about the result and your explanation (or expectation) about the result.

Submission: the result of this section will be a file `robot2.pl` with your representation in PROLOG and the answers to the questions as comments.

3.3 Planning (tricky)

For this third section, we consider the complete world. First of all, copy `robot2.pl` in a new file `robot3.pl`. For each step, TEST YOUR PROGRAM by making several queries. The purpose of this section is to make a deductive planning system for the robot.

Step 8: Write a predicate `serviceDone(S)` which says that in the situation `S`, every patient is not hungry and is not thirsty.

The purpose of a planning system is to provide at least one plan. The problem is that there is an infinite number of solutions. We want to restrict our search for plans with limited number of actions.

Step 9: Write a predicate `size(S,N)` which says that the number of actions in `S` is `N`. Write a predicate `plan(S,N)` which says that the situation has `N` actions and that the service is done.

Step 10: Write a predicate `goal(S,N)` which says that `S` is a plan with `N` actions at the most.

Example: we consider the following initial situation: "The robot is in the corridor, every patient is hungry but only the patients of room 2,3 are thirsty". Ask the following query: `goal(S,10).` (be careful, the computation can take several minutes).

Submission: the result of this section will be a file `robot3.pl` with your representation in PROLOG. If the query `goal(S,10).` does not work, you can provide tests where `S` is an instantiated term and the answer of the query is yes.

4 Advises

- A PROLOG program is always short. My own solution `robot3.pl` is less than 80 lines! (comments not included)
- Don't forget the order of the rules in the file is important (PROLOG triggers the rules from the top to the bottom). The order of the premises in a rule is also important (PROLOG unifies the predicate from the left first and propagates the result to the predicates on the right).
- This program is simple, no need of the cut operator (for a basic solution), no need of lists.
- Be careful, a noun beginning with a captial letter is always considered as a variable.
- The not operator is denoted `\+` in SWI-PROLOG. Try to avoid it if it is possible. If you feel the need of writing something like `\+ pred(X)`, try to consider the predicate `notpred(X)`.

  In order to say `\+ T1 = T2` you can also say `T1 \= T2`. If you use `T1 \= T2`, be sure that SWI-PROLOG will trigger it when the terms are fully instantiated.

  Example:
  if we suppose that `pred2(X,T1), pred3(Y,T2)` will unify `X` and `Y` with some ground terms then
  ```
  pred(X,Y) :- pred2(X,T1), pred3(Y,T2), pred4(T1,T2), X \= Y.
  ```
  is completely different from
  ```
  pred(X,Y) :- X \= Y, pred2(X,T1), pred3(Y,T2), pred4(T1,T2).
  ```
  from the unification point of view (backward chaining).
- Write:
  ```
  pred(toto(X)) :- pred(X).
  pred(titi(X)) :- pred(X).
  ```
  instead of
  ```
  pred(X) :- X = toto(Y), pred(Y); X = titi(Y), pred(Y).
  ```

Have fun!