CASL Single Lecture Notes

SLN - AIB - 05

2006-12-05

Cognitive Autonomous Systems Laboratory
Department of Computer and Information Science
Linköping university, Sweden

Erik Sandewall

Planning Using the Situation Calculus

Subject Area: Artificial Intelligence Basic

Date of lecture: 2006-11-09

"Single Lecture Notes" are notes corresponding to one lecture.

Related information can be obtained via the following WWW pages:

Linköping university:	http://www.liu.se/
This course:	http://www.ida.liu.se/~TDDA23/
CASL Group:	http://www.ida.liu.se/ext/casl/
The author:	http://www.ida.liu.se/~erisa/

1 Introduction

Given: histories of the world ('robotic histories'), characterized by a number of state variables as functions of time. Each state variable may be discrete (take one of a finite number of values) or piecewise continuous (continuous except for a limited number of discontinuities).

We have previously defined a *situation* in such a history and for a given set of state variables as a period of time during which:

- if all state variables are discrete: they keep the same value within the interval
- if piecewise continuous state variables are also included: there is no discontinuity within the interval. (Change of value of a discrete variable counts as a discontinuity).

Previously we characterized histories using the predicate H (also written Holds) where $H(t, f, v)$ says that the value of state variable f at time t is v.

State variables are also called fluents, or features.

Situation calculus (sitcalc for short) is an alternative approach that is designed for the case of discrete fluents (not obvious how to generalize to continuous ones) and which works as follows.

There is a predicate $H(s, f, v)$ that says that the fluent f has the value v in situation s. If v can only be 'true' or 'false' then one writes $H(s, f, true)$ as $H(s, f)$.

Actions are thought of as transitions from one situation to a successor. Each kind of action is represented as a function with one situation argument, and possibly other arguments, that has a new situation as its value. The value of $a(s)$ is intended as the new situation that is obtained as a successor of situation s by performing the action a.

In some variants of the sitcalc one writes $do(a, s)$ instead of $a(s)$.

In its basic form this approach has a number of limitations with respect to expressivity. Some of them can be removed by additional conventions. We shall return to that later, but first show the situation calculus approach at work.

An example

Like in some of the earlier lecture notes in this series, we shall write formulas in `typewriter font`, with the following, natural conventions:

```
->    implication
&     and
v     or
-     not
```

Consider a situation domain with one single fluent, called P, that specifies the position of the agent. The value of P shall be either of the following:

```
H    Home
LT   Linkoping travel center
SC   Stockholm central railway station
SS   Stockholm/Skavsta airport
SA   Stockholm/Arlanda airport
LS   London/Stanstead airport
LH   London/Heathrow airport
```

In addition there is the following base information:

```
H(s0, P, H)     where s0 is the initial situation
                (this is a standard convention in sitcalc)
Near(LS, London)
Near(LH, London)
```

and the following statement that we wish to prove:

```
∃ x ∃ s [ H(s, P, x) & Near(x,London) ]
```

For example, if we are able to prove

```
H( flyto-ls(busto-ss(walkto-lt(s0))), P, LS ) & Near(LS,London)
```

then we are done. The second half (the Near-expression) is already one of the axioms, so it is only required to prove the first part. It constitutes the plan, with the understanding that

```
walkto-lt(s)    is the situation that arises if one is in
                situation s and changes it by walking to LT
busto-ss(s)     is the situation that arises if one is in s
                and changes it by taking the bus to SS
flyto-ls(s)     is the situation that arises if one is in s
                and changes it by flying to LS
```

However these actions have preconditions: in order to do flyto-ls one has to be at SS, and so on. The preconditions and the effects are expressed as follows:

```
H(s,P,H)   ->  H(walkto-lt(s), P, LT)
H(s,P,LT)  ->  H(busto-ss(s), P, SS)
H(s,P,LT)  ->  H(trainto-sc(s), P, SC)
H(s,P,SC)  ->  H(busto-sa(s), P, SA)
H(s,P,SC)  ->  H(trainto-sa(s), P, SA)
H(s,P,SS)  ->  H(flyto-ls(s), P, LS)
H(s,P,SA)  ->  H(flyto-lh(s), P, LH)
```

with the obvious meaning for each of these. For example, the last axioms says (is intended to say) that if you are at Stockholm Arlanda airport and perform the action of flying to London Heathrow then you will be at London Heathrow.

Somewhat more realistically one would wish to make the destination an argument, for example

```
H(s,P,d)  ->  H(flyto(s,d), P, d)
```

but this requires the introduction of additional preconditions so we keep the first representation for the purpose of showing how deductions are made.

To prove the desired conclusion it is negated and combined with the axioms on clause form:

```
     -H(s, P, x) v -Near(x,London)

 1   H(s0, P, H)
 2   Near(LS, London)
 3   Near(LH, London)

 4  -H(s,P,H)   v  H(walkto-lt(s), P, LT)
 5  -H(s,P,LT)  v  H(busto-ss(s), P, SS)
 6  -H(s,P,LT)  v  H(trainto-sc(s), P, SC)
 7  -H(s,P,SC)  v  H(busto-sa(s), P, SA)
 8  -H(s,P,SC)  v  H(trainto-sa(s), P, SA)
 9  -H(s,P,SS)  v  H(flyto-ls(s), P, LS)
10  -H(s,P,SA)  v  H(flyto-lh(s), P, LH)
```

However, before we start the deduction process, using resolution, we also add an auxiliary literal called an answer literal to the negated desired conclusion, so that instead of the above one writes

```
11  -H(s, P, x) v -Near(x,London) v Answer(s)
```

This is a pure technicality and its purpose will soon become clear.

Now we can use the resolution operator, in ways similar to what we did in the previous lecture:

```
12  -H(s, P, LS) v Answer(s)                                   2,11
                                 (substituting LS for x in 11)

13  -H(s, P, SS) v Answer(flyto-ls(s))                         12,9
                                 (substituting flyto-ls(s) for s)

14  -H(s, P, LT) v Answer(flyto-ls(busto-ss(s)))               13,5

15  -H(s, P, H) v Answer(flyto-ls(busto-ss(walkto-lt(s))))     14,4

16  Answer(flyto-ls(busto-ss(walkto-lt(s0))))                  15,1
```

The final clause contains the answer: given the situation `s0`, do the actions `walkto-lt`, `busto-ss`, and `flyto-ls` in succession. `Answer` Remember now that the `Answer` literal was added technically. If we do the proof without it then clause 16 is the contradiction (empty clause) since 1 and 15 then are each other's negation (after substitution). In other words, the desired conclusion of the existence of a plan has been obtained, but as a side-effect and from the `Answer` literal we obtain the plan itself. We do not only prove the existence of a plan, we also construct it.

Actually the method of the `Answer` predicate is oldfashioned, and modern systems achieve the same purpose by extracting the solution from the proof itself, but that method is more difficult to follow. We use the `Answer` predicate here because it is very easy to understand.

Q-A and Discussion

1. If one has more than one fluent, then what are the effects on them (compare the Yale shooting problem)?

   Answer: You have to use some of the methods from the previous lecture, that is, use frame axioms, reverse frame axioms, or the like. However, systematic methods for doing this have been developed in the sitcalc framework.

2. What are the effects of an action if its preconditions are not satisfied?

   Answer: If the effects of the action are conditional on some aspect of the starting situation then you simply write several effect axioms, one for each case. For those cases where there is no effect axiom (which happens in the above example) then the answer for the previous question applies.

3. How do I represent the effects of nondeterministic actions?

   Answer: In basic sitcalc you cannot do that.

4. The proof above seems to determine the plan in reverse order, that is, it first finds the action `flyto-ls`, and then proceeds backwards. Is this always the case?

   Answer: Not necessarily, since you can do the resolutions in several different orders and get to the same result. However if you use the rule-of-thumb of always trying to use a clause containing the Answer predicate as one of the two clauses to resolve (which is a reasonable strategy), then this effect is obtained.

5. What about other restrictions on the expressivity of this method?

   Answer: No concurrent actions, no continuous state variables, not well suited for representing time, among other things.

6. What does "not well suited for representing time" mean?

   Answer: well, you can introduce fluents that obtain the starting time and the ending time of a situation as metric properties, but since actions have to be deterministic you will be bound to actions always taking the same number of time units, which is not realistic.

7. In the representation with H and D and time on the time axis as an argument, which we used in previous lectures, each interpretation of the logic formulas is a history of the world. What are the interpretations of sitcalc formulas?

   Answer: Not one timeline, but a tree of situations with `s0` at the root, and successors formed using the various actions which are situation-valued functions. In other words, all possible action sequences are included in one and the same interpretation.

8. Suppose I have the action `opendoor(s)` with the effect of opening a particular door, and `closedoor(s)` that closes the door. What is the relation between `s` and `closedoor(opendoor(s))` if the door is closed in situation `s`?

   Answer: different varieties of sitcalc take different stands on that, but in the major variant (Toronto sitcalc) they are considered as different situations although of course they are such that every fluent has the same value in one as in the other. In Toronto sitcalc, situations are isomorphic to (correspond 1-1 to) finite sequences of actions.

Some other variants of sitcalc consider situations to be the same as states of the world, so that each mapping from fluents to their values *is* a situation.