Situation Calculus

CS 579

September 30, 2004

1 Why Situation Calculus or Logic

Modeling dynamic domains is to

- provide a mathematical model of the domain
- take into consideration the dynamical behavior of opbjects in the domains.

Objects in a dynamic domain: environment and agents that can execute actions and change the environment. Agents can be of different types: reactive, deliberative, or hybrid.

A theory of dynamical systems needs to address

- the causal relationship between actions and their effects
- the executability condition of actions
- the exogenous actions
- the complex actions and procedures
- the concurrency between actions
- the probabilistic action occurrences and actions' effects
- the problem of determining what to do/when to do what
- the continuous property of processes
- the problem of how to update the belief of an agent when it executes a knowledge producing action
- the problem of unreliable hardware components (sensors etc.)
- the problem of how to belief/desire/intention of an agent affects its behavior
- the planning problem
- the problem of execution a course of actions and monitor its execution (to discover actions' failure and to recover from failure)
- the discrete/continuous time
- ...

In this course, we will use situation calculus to model dynamic domains. We make the following assumptions:

- there is only one agent
- there is no exogenous action

- reliable actions (actions do not fail, outcomes of action execution are predicatble)
- there is no probablistic actions (actions either occur or do not occur)
- no concurrency
- agents have complete knowledge about the environment and capabilities
- reasoning is done by deduction

In situation calculus, each domain is represented as a *set of logical formulas* that specify

- the actions and their effects,
- the executability conditions of actions, and
- the initial situation

The advantages of using logic:

- action behavior can be predicted using deduction,
- domain specification's properties such as the correctness of the specification can be formally proven, and
- in several instances, the specification is also an implementation of the system.

2 Situation Calculus – Formal Definitions

The main concepts of situation calculus:

- **Fluents**: properties of the domain whose value changed over time. Examples of fluent:
  1. location of a person
  2. temperature of a room
  3. speed of a car
  4. the age of a person
- **Situations**: a particular "snapshot" of the world. We require that each situation is the result of the environment's evolution from the initial situation. In this sense, we can view each situation as a sequence of actions. Examples of situation:
  1. 'at 5pm, 9/29/04, Son is in the room SH124',
  2. 'at 9pm, 9/29/04, the speed of Son's car is 0mph'
- **Actions:** the actions that agents/environments execute to change the world. Each action has its own effects and preconditions.

A **situation calculus theory** consists of

- A set of fluents
- A set of actions
- Situations, with s_0 is a constant denoting the initial situation
- Function *do* that maps a pair of an action and a situation into a situation; $do(a, s)$ denotes the successor situation to s resulting from executing action a

- Predicate *holds* whose first parameter is a fluent and second parameter is a situation
- A set of axioms about the initial situation (what is true/false in the initial situation?)
- A set of axioms that describes the effects of actions
- A set of axioms that describes the precondition of actions; for each action a, the theory consists of one formula of the form $Poss(a, s) \leftrightarrow Holds(\varphi, s)$ where φ is a fluent formula.

A few reminders:

The set of situation terms consists of term of the form

$$do(A_n, do(A_{n-1}, \ldots, do(A_1, S)))$$

where A_j's are variables representing actions and S is a variable representing a situation or s_0. This will not be true if we do not have the following axioms.

We define a binary relation $\sqsubset$ over the set of siutations. Intuitively, $S \sqsubset S'$ means that S precedes S'. The following axioms are called foundational axioms of the situation calculus:

- $do(A_1, S_1) = do(A_2, S_2) \rightarrow A_1 = A_2 \wedge S_1 = S_2$
- $\forall p(p(s_0) \wedge (\forall A, S)[p(S) \rightarrow p(do(A, S))] \rightarrow (\forall S)(p(S)))$
- $\neg S \sqsubset s_0$
- $S \sqsubset do(A, S') \equiv S \sqsubseteq S'$

The first axiom that each situation is unique. The second axiom is a second order axiom that minimizes the set of possible situations to the smallest set of situations containing s_0 and closed under the application of the *do* function to an action and a situation. This forces any model of the domain to contain only situation terms of the form we discuss above.

3 Illustration – The Rail Robot

Let us consider the domain in the following picture.

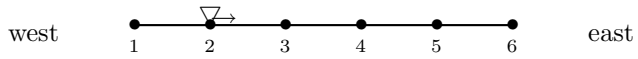

Figure 1: The Rail Car Domain – $\bigtriangledown$ denotes the car

The picture shows a railway with 6 stations. We assume that there is a car (or robot) that can move forward or turning 180 degree. It is not allowed to move past the station 6 or 1.

The only object (that we are interested in) in this domain is the car. The actions of the car are

1. *forward* – this will move the car to the 'next' station;
2. *turn* – this will change the heading of the car from *east* to *west* and vice versa.

The fluents of the rail robot domain are

1. *heading(X)* – the current heading of the car is X, where X can be either *east* or *west*;
2. *at(S)* – the car is at station S, where S can be 1, 2, 3, 4, 5, or 6.

We have that

$$heading(west) \leftrightarrow \neg heading(east)$$

(or $\neg heading(west) \leftrightarrow heading(east)$) and

$$at(N) \leftrightarrow \neg at(M) \text{ for } M \neq N$$

which means that some properties of the world can be automatically deduced when we know about other properties. We can describe the initial situation by

$$holds(at(2), s_0)$$
$$holds(heading(east), s_0)$$

Precondition axioms: Before an action is executed, it requires that some conditions are true. This condition is called the action's *precondition*. For this domain, we have only one requirement that the car does not move past the node 6 or node 1. This means that it cannot move forward when it is at node 6 and the heading is east or is at node 1 and the heading is west. We represent this by

$$\begin{aligned} poss(forward, S) &\leftrightarrow \neg((holds(at(6), S) \wedge holds(heading(east), S)) \vee \\ &\quad (holds(at(1), S) \wedge holds(heading(west), S))) \quad (1) \\ poss(turn, S) &\leftrightarrow true \quad (2) \end{aligned}$$

Effect axioms: We can represen the effects of the actions by

$$\begin{aligned} poss(forward, S) &\rightarrow ((holds(at(N), S) \wedge holds(heading(west), S)) \rightarrow holds(at(N-1), do(forward, S))) \vee \\ &\quad ((holds(at(N), S) \wedge holds(heading(east), S)) \rightarrow holds(at(N+1), do(forward, S))) \\ poss(turn, S) &\rightarrow (holds(heading(west), S) \rightarrow holds(heading(east), do(turn, S))) \vee \\ &\quad (holds(heading(east), S) \rightarrow holds(heading(west), do(turn, S))) \vee \end{aligned}$$

We could add the frame axioms in order to complete the representation of the domain. Instead of doing that, we write donw the successor state axioms for the fluents:

Successor state axioms: The general formula of the sussesor state axiom is of the form

$$holds(F, do(A, S)) \leftrightarrow \gamma_f^+(A, S) \vee (holds(A, S) \wedge \neg\gamma_f^-(A, S)) \quad (3)$$

So, for each of the fluent f and action a, we need to find $\gamma_f^+(a, S)$ and $\gamma_f^-(a, S)$. Here, for the fluent $at(N)$ and the action $forward$, we have

- $\gamma_{at(N)}^+(forward, S) = poss(forward, S) \wedge (holds(at(N+1), S) \wedge holds(heading(west, S)) \vee (holds(at(N-1), S) \wedge holds(heading(east, S)))$, i.e., if $\gamma_{at(N)}^+(forward, S)$ is true then $holds(at(N), do(forward, S))$ is true.
- $\gamma_{at(N)}^-(forward, S) = poss(forward, S) \wedge holds(at(N), S)$, i.e., if $\gamma_{at(N)}^-(forward, S)$ is true then $\neg holds(at(N), do(forward, S))$ is true.

This gives us the following

$$\begin{aligned} holds(at(N), do(forward, S)) &\leftrightarrow poss(forward, S) \wedge \\ &\quad (\\ &\quad (holds(at(N+1), S) \wedge holds(heading(west, S)) \vee \\ &\quad (holds(at(N-1), S) \wedge holds(heading(east, S)) \\ &\quad) \\ &\quad \vee \\ &\quad (holds(at(N), S) \wedge \neg(holds(at(N), S) \wedge poss(forward, S))) \end{aligned}$$

Simplify this, we get

$$\begin{aligned} holds(at(N), do(forward, S)) \quad \leftrightarrow \quad & poss(forward, S) \wedge \\ & (\\ & (holds(at(N+1), S) \wedge holds(heading(west, S)) \vee \\ & (holds(at(N-1), S) \wedge holds(heading(east, S)) \\ &) \\ & \vee \\ & (holds(at(N), S) \wedge \neg poss(forward, S)) \end{aligned}$$

in DNF form, it is

$$\begin{aligned} holds(at(N), do(forward, S)) \quad \leftrightarrow \quad & [poss(forward, S) \wedge holds(at(N+1), S) \wedge holds(heading(east, S)) \wedge N < 6] \vee \\ & [poss(forward, S) \wedge holds(at(N-1), S) \wedge holds(heading(west, S)) \wedge N > 1] \vee \\ & (holds(at(N), S) \wedge \neg poss(forward, S)) \end{aligned}$$

Similarly, we can derive the other successor state axioms that will be needed.

We call the theory T_R. The next section is about the block world, a well know domain in AI.

4 Another Example — The Block World

Let us consider the block world in the following picture.

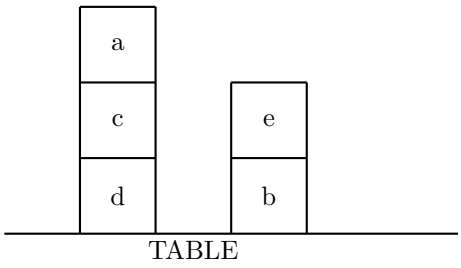

Figure 2: A Block World

We assume that the world consists of the above blocks and an agent who can pick up block and put it down on the table or on top of other blocks which are clear.

Objects of the block world are the blocks, the table, and the agent. The blocks are described by their names, *a,b,c,....* The table will be denoted by *t*. Since there is only one agent and no actions of the environment, we do not need to have a constant to denote the agent.

We will use the predicate $block(X)$ to denote that object X is a block.

The actions of the agents are:

1. $pickup(X)$ - pickup block X,
2. $put(X, Y)$ - put block X on top of block Y, and
3. $put(X, t)$ - put block X on the table.

The fluents of the block world are:

1. $on(X, Y)$ - block X is on the block Y,

2. $on(X, t)$ - block X is on the table,

3. $holding(X)$ - (the agent) is holding block X, and

4. $clear(X)$ - block X is clear.

Here we have that

$$clear(X) \leftrightarrow \neg\forall Y(block(Y) \wedge \neg on(Y, X))$$

In what follows, we will use different letters to denote different objects and omit the inequality axioms in writing the axioms of the domains.

The initial situation can be described by the following axioms

$$holds(on(a, c), s_0) \tag{4}$$
$$holds(on(c, d), s_0) \tag{5}$$
$$holds(on(d, t), s_0) \tag{6}$$
$$holds(on(e, b), s_0) \tag{7}$$
$$holds(on(b, t), s_0) \tag{8}$$
$$holds(clear(a), s_0) \tag{9}$$
$$holds(clear(e), s_0) \tag{10}$$

Note: we do not specify *what does not hold in* s_0. We will do it by stating that what is not specified to be true in s_0 will be considered as false (the *closed world assumption*.

Now, let us describe the set of precondition axioms and the set of effect axioms.

Precondition axioms: Before an action is executed, it requires that some conditions are true. This condition is called the action's *precondition*. For example, to put down a block X on top of block Y, the agent must hold X and Y must be clear. Another example is that to pick up a block X, the agent must not hold another block and X must be clear etc... We represent these by

$$poss(pickup(X), S) \rightarrow \forall Y(holds(\neg holding(Y), S)) \wedge holds(clear(X), S) \tag{11}$$
$$poss(put(X, Y), S) \rightarrow holds(holding(X), S) \wedge holds(clear(Y), S) \tag{12}$$
$$poss(put(X, t), S) \rightarrow holds(holding(X), S) \tag{13}$$

Problem The above formulas do not allow us to prove when an action, say $pickup(X)$, is possible. I.e., when $poss(pickup(X), S)$ is true. Remember how to prove whether a literal is true?

Can we reverse the axioms (change $\rightarrow$ with $\leftarrow$) and then use them to prove when $poss(pickup(X), S)$ is true? No, this will not work since it is not correct! There might be conditions which we have not considered? For example, the block can be too heavy, the agent might be tired etc... So, we need to specify all of these conditions, called *qualifications*. But there are infinitely many conditions like this?

The above is called the *qualification problem*. It calls for the need for *nonmonotonic reasoning* in AI.

To overcome the qualification problem, we make the assumption that the antecedent of the above formula represents the sufficient and necessary conditions for the action to be executed. The *final* action precondition axioms for the above three actions are:

$$poss(pickup(X), S) \leftrightarrow \forall Y(holds(\neg holding(Y), S)) \wedge holds(clear(X), S) \tag{14}$$
$$poss(put(X, Y), S) \leftrightarrow holds(holding(X), S) \wedge holds(clear(Y), S) \tag{15}$$
$$poss(put(X, t), S) \leftrightarrow holds(holding(X), S) \tag{16}$$

We will use the above axioms instead of the axioms (11)-(13).

Given the above axioms and some axioms about the , we can now deduce whether the action $pickup(a)$ is possible in the initial situation, i.e., $poss(pickup(a), s_0)$ is true. How?

Effect axioms: Actions have effects. For example, if the agent pickups a block he will hold it in the successor situation.

$$poss(pickup(X), S) \rightarrow holds(holding(X), do(pickup(X), S)) \tag{17}$$
$$poss(put(X, Y), S) \rightarrow holds(on(X, Y), do(put(X, Y), S)) \tag{18}$$
$$poss(put(X, t), S) \rightarrow holds(on(X, t), do(put(X, t), S)) \tag{19}$$
$$poss(pickup(X), S) \rightarrow [holds(on(X, Y), S) \rightarrow holds(\neg on(X, Y), do(pickup(X), S))] \tag{20}$$

But, the above equations only represent the 'positive effects' of actions (what becomes true) and 'negative effects' of actions (what becomes false) as well. In (20), for example, if the agent pickups block X which is on a block Y in the situation s then $on(X, Y)$ will not hold in $do(pickup(X), S)$; or if the agent puts block X on a block Y in the situation s then $clear(Y)$ will not hold in $do(put(X, Y), S)$, etc. **Try to complete the set of effect axioms.**

Furthermore, there are fluents whose values do not change after the execution of an action. For example, if X is on Y in situation S ($holds(on(X, Y), S)$ is true) and the agent puts block Z on top of X, then X is still on Y in situation $do(put(z, X), S)$ ($holds(on(X, Y), do(put(Z, X), S))$ is true). Representing the unchange effects of actions require axioms of the form

$$holds(on(X, Y), S) \rightarrow holds(on(X, Y), do(pickup(Z), S))$$

These axioms are called *frame axioms*.

If we have n actions, m fluents, we would need $2 \times n \times m$ frame axioms. This is too many!!!

Representing frame axioms in a compact way is a challenge for quite sometime. To date, there are many solutions to the frame problems. Under reasonable assumptions, we can write, for each fluent F, one *successor state axiom* in the following form

$$holds(F, do(A, S)) \leftrightarrow \gamma_f^+(A, S) \vee (holds(A, S) \wedge \gamma_f^-(A, S)) \tag{21}$$

where $\gamma_f^+(A, S)$ (resp. $\gamma_f^-(A, S)$) summarizes the conditions for the fluent F to become true (resp. false).

For example, in the block world we have

$$holds(on(X, Y), do(A, S)) \leftrightarrow \gamma^+_{on(X,Y)}(A, S) \vee (holds(on(X, Y), S) \wedge \gamma^-_{on(X,Y)}(A, S)) \tag{22}$$

where

$\gamma^+_{on(X,Y)}(A, S) = (A = put(X, Y)) \wedge poss(A, S)$ and

$\gamma^-_{on(X,Y)}(A, S) = (A \neq pickup(X)) \wedge poss(A, S)$.

Because of the situation $do(A, S)$ is valid only when A is executable in S, i.e., $poss(A, S)$ holds, we can write the above axiom as follows:

$$poss(A, S) \rightarrow (holds(on(X, Y), do(A, S)) \leftrightarrow A = put(X, Y) \vee (holds(on(X, Y), S) \wedge A \neq pickup(X))) \tag{23}$$

The axioms for other fluents are listed below:

For $on(X, t)$:

$$poss(A, S) \rightarrow (holds(on(X, t), do(A, S)) \leftrightarrow A = put(X, t) \vee (holds(on(X, t), S) \wedge A \neq pickup(X))) \tag{24}$$

For $holding(X)$:

$$poss(A,S) \rightarrow (holds(holding(X), do(A,S)) \leftrightarrow A = pickup(X) \vee \\ (holds(holding(X), S) \wedge A \neq put(X,Y) \wedge A \neq put(X,t))) \tag{25}$$

For $clear(X)$:

$$poss(A,S) \rightarrow (holds(clear(X), do(A,S)) \leftrightarrow (A = pickup(Y) \wedge holds(on(Y,X), S)) \vee \\ (holds(clear(X), S) \wedge A \neq put(Y,X)) \wedge A \neq pickup(X)) \tag{26}$$

Summary: The situation calculus theory for the block world domain consists of the axioms (4)-(10) (initial situation), (14)-(16) (executability condition), and (23)-(26). We will refer to this theory as T_B.

5 Deduction in T_R and T_B

Read this section if you are interested in 'proving' things.

• For the rail car domain, we can show that

$$T_R \models holds(at(3), do(forward, s_0))$$

as follows. From the last axiom of T_R, we know that the above can be proved if

$$T_R \models poss(forward, s_0) \wedge holds(at(N-1), s_0) \wedge holds(heading(east, s_0))$$

holds. Because $holds(at(2), s0)$ and $holds(heading(east), s_0)$ hold, this means that we need to prove that

$$T_R \models poss(forward, s_0)$$

From the equation (1) and the fact that $holds(at(2), s0)$ and $holds(heading(east), s_0)$ are both true, we conclude that $T_R \models poss(forward, s_0)$ holds, i.e., $T_R \models holds(at(2), do(forward, s_0))$.

• For the block world domain, we can show that

$T_B \models poss(pickup(a), s_0)$. (P1)

To prove this, we note that (from (14)) that

$$poss(pickup(a), s_0) \leftrightarrow \forall Y(holds(\neg holding(Y), s_0)) \wedge holds(clear(a), s_0)$$

Because $T_B \models \forall Y(holds(\neg holding(Y), s_0))$ and $T_B \models holds(clear(a), s_0)$ (from (4)-(10) and CWA), we have the conclusion.

• $T_B \models holds(holding(a), do(pickup(a), s_0))$. (P2)

To prove this, we need to use the successor state axiom for the fluent $holding(a)$. This is the axiom (25). Because we have proved that $T_B \models poss(pickup(a), s_0)$, all we need to show now that

$$T_B \models pickup(a) = pickup(a) \vee (holds(holding(a), s_0) \wedge pickup(a) \neq put(a,Y) \wedge pickup(a) \neq put(a,t)).$$

Clearly this holds because of the equality $pickup(a) = pickup(a)$. Thus, we can conclude that (P2) holds.

• $T_B \models \neg holds(clear(a), do(pickup(a), s_0))$. (P3)

Similar to the proof of (P2), we need to show that

$$T_B \not\models (pickup(a) = pickup(Y) \wedge holds(on(Y,a), s_0)) \vee \\ (holds(clear(a), s_0) \wedge pickup(a) \neq put(Y,a) \wedge pickup(a) \neq pickup(a)).$$

Clearly, $T_B \not\models pickup(a) = pickup(Y) \wedge holds(on(Y,a), s_0)$ because there is $on(a,a)$ is not tru in s_0. Furthermore, $T_B \not\models holds(clear(a), s_0) \wedge pickup(a) \neq put(Y,a) \wedge pickup(a) \neq pickup(a)$ because of the last inequality. Therefore, we can conclude that (P3) is true.

6 Implementation in Prolog

6.1 The Railway Domain

```
%
% auxiliary predicates
% defining the stations and directions (i.e., defining constants)
%
station(X):- member(X,[1,2,3,4,5,6]).
direction(Y):- member(Y, [east, west]).

% -- prim_fluent(?Fluent): for each primitive fluent
% -- prim_action(?Action): for each primitive action

prim_fluent(at(X)):- station(X).        % at(X) is a fluent if X is a station
prim_fluent(heading(Y)):- direction(Y). % current heading is Y

prim_action(forward).
prim_action(turn).

% specify action preconditions

poss(forward, S):-
      (holds(at(X), S), holds(heading(east), S), X < 6);
      (holds(at(X), S), holds(heading(west), S), X > 6).

poss(turn, S).

% successor state axiom

holds(at(N), do(forward, S)):-
      station(N),
      poss(forward, S),
        N1 is N+1, holds(at(N1), S), holds(heading(west), S).

holds(at(N), do(forward, S)):-
      station(N),
      poss(forward, S),
        N1 is N-1, holds(at(N1), S), holds(heading(east), S).

holds(at(N), do(turn, S)):- station(N),
      poss(turn, S), holds(at(N), S).

holds(heading(east), do(turn, S)):-
      poss(turn, S), holds(heading(west), S).

holds(heading(west), do(turn, S)):-
      poss(turn, S), holds(heading(east), S).

holds(heading(D), do(forward, S)):- direction(D),
      poss(forward, S), holds(heading(D), S).

% initial situation
```

```
holds(at(2), s0).
holds(heading(east), s0).
```

It should be said that we have removed the third disjunct in the

```
holds(at(N), do(forward, S)):-
      holds(at(N), S), not poss(forward, S).
```

This code is available as **car.pl** from the classnote URL. It allows you to ask questions like:

- ? `poss(forward, s0)` – Yes.
- ? `holds(at(3), do(forward,s0))` – Yes
- ? `holds(heading(west), do(forward, s0))` – No
- ? `holds(at(4), X)` – X = do(forward, do(forward, s0))

6.2 The Block World Domain

First, we need to represent the domain in Prolog. One possiblity is the following:

```
% Specifying the objects

bl(a).
bl(b).
bl(c).
bl(d).
bl(e).
table(t).

% defining primitive fluents

prim_fluent(on(X,Y)):-
   bl(X), bl(Y), neq(X,Y).

prim_fluent(on(X,t)):-
   bl(X).

prim_fluent(clear(X)):-
   bl(X).

prim_fluent(holding(X)):-
   bl(X).

prim_fluent(empty).   %% added

% defining actions

prim_action(pickup(X)):-
   bl(X).

prim_action(put(X,Y)):-
   bl(X), bl(Y), neq(X,Y).
```

```
prim_action(put(X,t)):-
   bl(X).

% excutability conditions

poss(pickup(X), S):-
   prim_action(pickup(X)),
   holds(clear(X), S),
   holds(empty, S).      %%% added

poss(put(X,Y), S) :-
   prim_action(put(X,Y)),
   holds(holding(X), S),
   holds(clear(Y), S).

poss(put(X,t), S) :-
   prim_action(put(X,t)),
   holds(holding(X), S).

% initial situation

holds(on(a,c), s0).
holds(on(c,d), s0).
holds(on(d,t), s0).
holds(on(e,b), s0).
holds(on(b,t), s0).
holds(clear(a), s0).
holds(clear(e), s0).

% SSA

holds(on(X,Y), do(A,S)):-
   prim_action(A), poss(A, S), A = put(X,Y);
   \+ A = pickup(X), holds(on(X,Y),S).

holds(on(X,t), do(A,S)):-
   prim_action(A), poss(A, S), A = put(X,t);
   \+ A = pickup(X), holds(on(X,t),S).

holds(holding(X), do(A,S)):-
   prim_action(A), poss(A,S), A = pickup(X);
   \+ A = put(X,t), \+ A = put(X,Y),
   holds(holding(X),S).

holds(clear(X), do(A,S)):-
   prim_action(A), poss(A,S),
   holds(on(Y,X),S),
   A = pickup(Y);
   \+ A = put(Y,X),
   \+ A = pickup(X),
   holds(clear(X),S).

holds(something, S):- bl(X), holds(holding(X), S).
```

```
holds(empty, S):- \+ holds(something, S).

% auxiliary

eq(X,X).

neq(X,Y):- not eq(X,Y).
```

Have a closer look and see if there any problem with the above implementation. Identify them!

We can now pose the following queries to the interpreter:

- `? poss(pickup(a), s0)` – Yes.
- `? holds(clear(a), do(pickup(a),s0))` – No.
- `? holds(clear(c), do(pickup(a),s0))` – Yes.