Knowledge Representation: Situation Calculus

Situation Calculus

The Situation Calculus is a first-order predicate language designed for representing and reasoning about dynamical worlds

https://en.wikipedia.org/wiki/Situation_calculus

Note: first-order predicate calculus is actually an infinite family of languages. The version of the situation calculus we will use in this course is a member of that family.

First introduced by John McCarthy in 1963

Developed further by the KR group at the University of Toronto: see papers/books by Ray Reiter, Hector Levesque

Situation Calculus (cont'd)

The basic elements of the calculus are:

- The ***actions*** that can be performed in the world
- The ***fluents*** that describe the state of the world
- The ***situations*** that represent courses of action

A domain is formalized by a number of axioms, namely:

- ***Action precondition axioms***, one for each action
- ***Successor state axioms***, one for each fluent
- Axioms describing the ***initial state*** of the world
- The ***foundational axioms*** of the situation calculus

Situation Calculus (cont'd)

The basic elements of the calculus:

- ***actions*** are things
- ***fluents*** are predicates that give state of the world which depends on situation
- ***situations*** are effectively lists of actions (read right to left)

A domain is formalized by a number of axioms, namely:

- ***Action precondition axioms:*** under what conditions is each action possible?
- ***Successor state axioms:*** what is true after an action occurrence?
- ***Initial State axioms:*** what is true before any action happens?
- ***Foundational Axioms:*** implicit for us -- we concentrate on the above

Axioms

What are axioms?

Axioms are the initial statements (logical formulae) that we make when we are representing a domain

What do I mean by a domain?

A domain is a part of the world that's relevant to the reasoning we want to do, for example

- In a block-stacking domain, we would write down everything we can about blocks, stacking, moving, putting down, and so
- If we want to represent a bigger subset of the world, there is more we need to write down to represent that world

Actions

We will concentrate on simple actions whose effects don't depend on time

The formalism works with complex/concurrent actions, and actions that depend on time, but we will keep things simple

Actions are the mechanism for aspects of our domain to change

Example actions:

pick_up(block(block1))

move_to(location(x),location(y))

put_down(block(block1))

Action Schema

Sometimes we represent a number of actions with a single expression called an *Action Schema.*

Blocks World Action Schema: move(Block,Src,Dst)

3 blocks, 4 positions in the blocks world of this course

Single expression representing all the individual actions -- all combinations of (Block,Src,Dst), of which there are 3x7x7:

move(block(b1),block(b1),block(b1))

move(block(b1),block(b1),block(b2))

move(block(b1),block(b1),block(b3))

move(block(b1),block(b1),position(p1)) …. etc …. etc …

Fluents

Fluents are predicates that take a situation argument

They are called fluents because they represent statements whose truth value changes (due to actions)

They take a situation (action history) as the last argument

Fluents are used to represent the situation-dependent state of the world (hence their last argument is a situation)

Examples of Fluents being used to make statements:

on(block(block1),block(block2),s)

holding(block(block3),s)

position(location(x),location(y),s)

Situations

It's tempting to think of a situation as a state, but in the modern versions of the situation calculus, a situation is

- an action history, or in other words, a course of action
- There is a distinguished function symbol $\mathbf{S_0}$ representing the action history of no action
- There is a distinguished function symbol **do(a,s)** representing the situation (action history) resulting from doing a "in" or equivalently "after" s
- Example situations
- do(pick_up(block(block1)),S_0)
- do(put_down(block(block1)),do(move(location(x),location(y)),do(pick_up(block(block1)),S_0)))

Situation observations

do(put_down(block(block1)),do(move(location(x),location(y)),do(pick_up(block(block1)),S_0)))

1. This represents a course of action consisting of three actions, in order: **pick_up(block(block1)), move(location(x),location(y)), put_down(block(block1))**
2. Similarity to peano number theory:

succ(succ(succ(0))) is the formal representation of 3

3. Similarity to prolog lists, using the dot (**.**) functor notation

. (put_down(block(block1)),.(move(location(x),location(y)),.(pick_up(block(block1)),[ ])))

or in prolog regular notation for lists

[put_down(block(block1)),move(location(x),location(y)),pick_up(block(block1))]

Situation observations (cont'd)

do(put_down(block(block1)),do(move(location(x),location(y)),do(pick_up(block(block1)),S_0)))

We can be really clever Prolog programmers and adopt the following convention for situations:

1. The special situation where nothing has happened, **S_0**, will be represented by the special Prolog atom **[]**
2. The function symbol **do** will be represented by the list functor
3. do(a,S_0) will be written in Prolog as [a] or [a|[]]
4. do(a,S) will be written in Prolog as [a|S]

The above situation becomes a Prolog list, read RIGHT to LEFT:

[put_down(block(block1)),move(location(x),location(y)),pick_up(block(block1))]

Action precondition axioms

For each action, we need to state up front where that action is possible and where it isn't possible

"where" here means: "after which courses of action?" or alternatively we could say "in which situations?"

General form of precondition axiom:

$$Poss(A(\vec{x}), s) \equiv \Phi(\vec{x}, s)$$

$A(\vec{x})$ is an action, where $\vec{x}$ represents all the arguments of the action

$\Phi(\vec{x}, s)$ is a logical formula involving fluents, characterizing the state where $A(\vec{x})$ is possible

Action Precondition axioms (cont'd)

Example Precondition axiom (next slide)

- move(x,y,z) denotes an action of moving Block x from Block y to Block z
- on(x,y,s) means that Block x is on Position or Block y in Situation S
- clear(x,s) means that Block x is clear in Situation s

Sitcalc precondition axiom

```
%Let's add comments to this code together

poss([move(Block,From,To)|S]):-
  block_exists(Block),
  clear(Block,S),
  (location_exists(To) ; block_exists(To)),
  Block \= To,
  clear(To,S),
  (location_exists(From);block_exists(From)),
  on(Block,From,S).
```

Action Precondition axioms observations

With situations, we observed that it is convenient to use Prolog's list notation:

- The empty list [] can represent the initial situation S_0
- List notation [A|S] can represent the situation do(a,s)

We make a similar observation that the Poss(a,s) predicate (**two arguments**):

```
poss(move(BlockA,BlockB),S) :- % two arguments for poss
        clear(BlockA,S), on(BlockA,BlockB,S), clear(BlockC,S).
```

could be equivalently written using Prolog list notation (**one argument**) as

```
poss([move(BlockA,BlockB)|S]) :-  % one argument for poss
        clear(BlockA,S), on(BlockA,BlockB,S), clear(BlockC,S).
```

Whether to use poss/2 or poss/1 is arbitrary, but we must be consistent!

Successor State Axioms

For each fluent, we need to state up front the conditions under which it becomes true, false, or remains unchanged

The form of a successor state axiom is

$$Poss(A(\vec{x}), s) \supset R(\vec{y}, do(A(\vec{x}), s)) \equiv$$
$$\gamma_R^+(\vec{y}, A(\vec{x}), s) \vee$$
$$R(\vec{y}, s) \wedge \neg\gamma_R^-(\vec{y}, A(\vec{x}), s)$$

where

$\gamma_R^+(\vec{y}, A(\vec{x}), s)$ represents the conditions under which $R(\vec{y}, do(A(\vec{x}), s))$ is true

$\gamma_R^-(\vec{y}, A(\vec{x}), s)$ represent the conditions under which $R(\vec{y}, do(A(\vec{x}), s))$ is false

Successor State Axioms (cont'd)

In English, we would read the Successor State Axiom as

If Action A is possible in s, then

R is true after performing A in s if and and only if

the conditions are such that A makes R become true

or

R was already true, and conditions are such that A does not make R false

Successor State Axioms

Successor State Axiom for on(X,Y,S):

$$Poss(A(\vec{x}), s) \supset on(BlockA, BlockC, do(A(\vec{x}), s)) \equiv$$
$$A(\vec{x}) = move(BlockA, BlockB, BlockC) \quad \vee$$
$$on(BlockA, BlockC, s) \wedge \neg\, A(\vec{x}) = move(BlockA, BlockC, Someblock)$$

where

$\gamma_R^+(\vec{y}, A(\vec{x}), s)$ represents the conditions under which $R(\vec{y}, do(A(\vec{x}), s))$ is true

$A(\vec{x}) = move(BlockA, BlockB, BlockC)$

$\gamma_R^-(\vec{y}, A(\vec{x}), s)$ represent the conditions under which $R(\vec{y}, do(A(\vec{x}), s))$ is false

$A(\vec{x}) = move(BlockA, BlockC, Someblock)$

SitCalc successor state axioms (Prolog Syntax)

```
% Let's add comments to this code together

clear(X,[move(_,X,_)|S]):- poss([move(_,X,_)|S]).
clear(X,[A|S]):-
  poss([A|S]),
  A \= move(_,_,X),
  clear(X,S).


on(X,Y,[move(X,Z,Y)|S]):- poss([move(X,Z,Y)|S]).
on(X,Y,[A|S]):-
  poss([A|S]),
  A \= move(X,Y,_),
  on(X,Y,S).
```

Axiomatizing a Domain

- Axiomatizing a Domain means that an axiomatizer (you?) writes down statements that define what is true about a domain.
- What is a Domain?
  - A Domain is a specific area or field of knowledge, expertise, or subject matter that an AI system or knowledge base is designed to understand and reason about
- Examples of domains:
  - Taxi domain: taxis pick up passengers and drop them off at destinations
  - Kitchen domain: a robot makes dinner in a kitchen
  - Cardiology domain: a doctor sees heart patients, interviews them, conducts tests, and diagnoses them

Steps to Axiomatize a Domain

In the following steps, the word "determine" implies "write down"

1. Understand the domain by reading about it, studying it, and thinking about it
2. Determine the set of fluents that are sufficient to represent a state in the domain
3. Determine the set of actions that effect (bring about) change in the state (fluent truth values)
4. Determine the precondition axioms for actions in terms of fluents
5. Determine the successor state axiom for each fluent
6. Determine the fluent values in the initial situation s0 (we use [] for s0).

Time to check your learning!

Let’s see how many key concepts from Situation Calculus you recall by answering the following questions!

In plain English, what is being specified when Precondition Axioms are written down for a domain?

In plain English, what is being specified when Successor State Axioms are written down for a domain?

What is meant by "domain" in the above two questions?