14.

Actions

Situation calculus

The situation calculus is a dialect of FOL for representing dynamically changing worlds in which all changes are the result of named actions.

There are two distinguished sorts of terms:

- actions, such as
  - put(x,y) put object x on top of object y
  - walk(loc) walk to location loc
  - pickup(r,x) robot r picks up object x
- situations, denoting possible world histories. A distinguished constant S_0 and function symbol do are used
  - S_0 the initial situation, before any actions have been performed
  - $do(a,s)$ the situation that results from doing action a in situation s

for example: $do(\text{put}(A,B),do(\text{put}(B,C),S_0))$ the situation that results from putting A on B after putting B on C in the initial situation

Fluents

Predicates or functions whose values may vary from situation to situation are called fluents.

These are written using predicate or function symbols whose last argument is a situation

for example: $\text{Holding}(r, x, s)$: robot r is holding object x in situation s

can have: $\neg\text{Holding}(r, x, s) \wedge \text{Holding}(r, x, do(\text{pickup}(r,x),s))$

the robot is not holding the object x in situation s, but is holding it in the situation that results from picking it up

Note: there is no distinguished "current" situation. A sentence can talk about many different situations, past, present, or future.

A distinguished predicate symbol $Poss(a,s)$ is used to state that a may be performed in situation s

for example: $Poss(\text{pickup}(r,x), S_0)$

it is possible for the robot r to pickup object x in the initial situation

This is the entire language.

Preconditions and effects

It is necessary to include in a KB not only facts about the initial situation, but also about world dynamics: what the actions do.

Actions typically have preconditions: what needs to be true for the action to be performed

- $Poss(\text{pickup}(r,x), s) \equiv \forall z.\ \neg\text{Holding}(r,z,s) \wedge \neg\text{Heavy}(x) \wedge \text{NextTo}(r,x,s)$

  a robot can pickup an object iff it is not holding anything, the object is not too heavy, and the robot is next to the object

  Note: free variables assumed to be universally quantified

- $Poss(\text{repair}(r,x), s) \equiv \text{HasGlue}(r,s) \wedge \text{Broken}(x,s)$

  it is possible to repair an object iff the object is broken and the robot has glue

Actions typically have effects: the fluents that change as the result of performing the action

- $\text{Fragile}(x) \supset \text{Broken}(x, do(\text{drop}(r,x),s))$

  dropping a fragile object causes it to break

- $\neg\text{Broken}(x, do(\text{repair}(r,x),s))$

  repairing an object causes it to be unbroken

The frame problem

To really know how the world works, it is also necessary to know what fluents are *unaffected* by performing an action.

- $\text{Colour}(x,c,s) \supset \text{Colour}(x, c, do(\text{drop}(r,x),s))$

  dropping an object does not change its colour

- $\neg\text{Broken}(x,s) \wedge [\, x \neq y \vee \neg\text{Fragile}(x) \,] \supset \neg\text{Broken}(x, do(\text{drop}(r,y),s)$

  not breaking things

These are sometimes called frame axioms.

Problem: need to know a vast number of such axioms. (Few actions affect the value of a given fluent; most leave it invariant.)

an object's colour is unaffected by picking things up, opening a door, using the phone, turning on a light, electing a new Prime Minister of Canada, *etc.*

The frame problem:

- in building KB, need to think of these $\sim 2 \times A \times F$ facts about what does not change
- the system needs to reason efficiently with them

What counts as a solution?

- Suppose the person responsible for building a KB has written down *all* the effect axioms

  for each fluent F and action A that can cause the truth value of F to change, an axiom of the form $[R(s) \supset \pm F(do(A,s))]$, where $R(s)$ is some condition on s

- We want a systematic procedure for generating all the frame axioms from these effect axioms
- If possible, we also want a *parsimonious* representation for them (since in their simplest form, there are too many)

Why do we want such a solution?

- frame axioms are necessary to reason about actions and are not entailed by the other axioms
- convenience for the KB builder
  - modularity: only add effect axioms
  - accuracy: no inadvertent omissions
- for theorizing about actions

The projection task

What can we do with the situation calculus?

We will see later that it can be used for planning.

A simpler job we can handle directly is called the projection task.

> Given a sequence of actions, determine what would be true in the situation that results from performing that sequence.

This can be formalized as follows:

> Suppose that $R(s)$ is a formula with a free situation variable s.
>
> To find out if $R(s)$ would be true after performing $\langle a_1,...,a_n \rangle$ in the initial situation, we determine whether or not

$$KB \models R(do(a_n,do(a_{n-1},...,do(a_1,S_0)...)))$$

For example, using the effect and frame axioms from before, it follows that $\neg \text{Broken}(\text{B},s)$ would hold after doing the sequence

$$\langle \text{pickup(A), pickup(B), drop(B), repair(B), drop(A)} \rangle$$

The legality task

The projection task above asks if a condition would hold after performing a sequence of actions, but not whether that sequence can in fact be properly executed.

We call a situation legal if it is the initial situation or the result of performing an action whose preconditions are satisfied starting in a legal situation.

The legality task is the task of determining whether a sequence of actions leads to a legal situation.

This can be formalized as follows:

> To find out if the sequence $\langle a_1,...,a_n \rangle$ can be legally performed in the initial situation, we determine whether or not
>
> $$KB \models Poss(a_i, do(a_{i-1},...,do(a_1,S_0)...))$$
>
> for every i such that $1 \leq i \leq n$.

Limitations of the situation calculus

This version of the situation calculus has a number of limitations:

- no time: cannot talk about how long actions take, or when they occur
- only known actions: no hidden exogenous actions, no unnamed events
- no concurrency: cannot talk about doing two actions at once
- only discrete situations: no continuous actions, like pushing an object from A to B.
- only hypotheticals: cannot say that an action has occurred or will occur
- only primitive actions: no actions made up of other parts, like conditionals or iterations

We will deal with the last of these below.

First we consider a simple solution to the frame problem ...

Normal form for effect axioms

Suppose there are two positive effect axioms for the fluent *Broken:*

$$\text{Fragile}(x) \supset \text{Broken}(x, do(\text{drop}(r,x), s))$$

$$\text{NextTo}(b,x,s) \supset \text{Broken}(x, do(\text{explode}(b), s))$$

These can be rewritten as

$$\exists r \{a = \text{drop}(r,x) \wedge \text{Fragile}(x)\} \vee \exists b \{a = \text{explode}(b) \wedge \text{NextTo}(b,x,s)\} \supset \text{Broken}(x, do(a,s))$$

Similarly, consider the negative effect axiom:

$$\neg\text{Broken}(x, do(\text{repair}(r,x), s))$$

which can be rewritten as

$$\exists r \{a = \text{repair}(r,x)\} \supset \neg\text{Broken}(x, do(a,s))$$

In general, for any fluent F, we can rewrite all the effect axioms as as two formulas of the form

$$P_F(\boldsymbol{x}, a, s) \supset F(\boldsymbol{x}, do(a,s)) \quad (1)$$

$$N_F(\boldsymbol{x}, a, s) \supset \neg F(\boldsymbol{x}, do(a,s)) \quad (2)$$

where $P_F(\boldsymbol{x}, a, s)$ and $N_F(\boldsymbol{x}, a, s)$ are formulas whose free variables are among the x_i, a, and s.

Explanation closure

Now make a completeness assumption regarding these effect axioms:

assume that (1) and (2) characterize *all* the conditions under which an action a changes the value of fluent F.

This can be formalized by explanation closure axioms:

$$\neg F(\mathbf{x}, s) \wedge F(\mathbf{x}, do(a,s)) \supset \mathrm{P}_F(\mathbf{x}, a, s) \qquad (3)$$

if F was false and was made true by doing action a then condition P_F must have been true

$$F(\mathbf{x}, s) \wedge \neg F(\mathbf{x}, do(a,s)) \supset \mathrm{N}_F(\mathbf{x}, a, s) \qquad (4)$$

if F was true and was made false by doing action a then condition N_F must have been true

These explanation closure axioms are in fact disguised versions of frame axioms!

$$\neg F(\mathbf{x}, s) \wedge \neg \mathrm{P}_F(\mathbf{x}, a, s) \supset \neg F(\mathbf{x}, do(a,s))$$

$$F(\mathbf{x}, s) \wedge \neg \mathrm{N}_F(\mathbf{x}, a, s) \supset F(\mathbf{x}, do(a,s))$$

Successor state axioms

Further assume that our KB entails the following

- integrity of the effect axioms: $\neg \exists \boldsymbol{x}, a, s.\ \mathrm{P}_F(\boldsymbol{x}, a, s) \wedge \mathrm{N}_F(\boldsymbol{x}, a, s)$
- unique names for actions:

$$A(x_1,\ldots,x_n) = A(y_1,\ldots,y_n) \supset (x_1=y_1) \wedge \ldots \wedge (x_n=y_n)$$
$$A(x_1,\ldots,x_n) \neq B(y_1,\ldots,y_m) \quad \text{where } A \text{ and } B \text{ are distinct}$$

Then it can be shown that KB entails that (1), (2), (3), and (4) together are logically equivalent to

$$F(\boldsymbol{x}, do(a,s)) \equiv \mathrm{P}_F(\boldsymbol{x}, a, s) \vee (F(\boldsymbol{x}, s) \wedge \neg \mathrm{N}_F(\boldsymbol{x}, a, s))$$

This is called the successor state axiom for F.

For example, the successor state axiom for the *Broken* fluent is:

$$\begin{aligned} &\mathrm{Broken}(x, do(a,s)) \equiv \\ &\quad \exists r\ \{a = \mathrm{drop}(r,x) \wedge \mathrm{Fragile}(x)\} \\ &\vee\ \exists b\ \{a = \mathrm{explode}(b) \wedge \mathrm{NextTo}(b,x,s)\} \\ &\vee\ \mathrm{Broken}(x, s) \wedge \neg \exists r\ \{a = \mathrm{repair}(r,x)\} \end{aligned}$$

An object x is broken after doing action a
iff
a is a dropping action and x is fragile,
or a is a bomb exploding
where x is next to the bomb,
or x was already broken and
a is not the action of repairing it

Note universal quantification: for *any* action a ...

A simple solution to the frame problem

This simple solution to the frame problem (due to Ray Reiter) yields the following axioms:

- one successor state axiom per fluent
- one precondition axiom per action
- unique name axioms for actions

Moreover, we do not get fewer axioms at the expense of prohibitively long ones

> the length of a successor state axioms is roughly proportional to the number of actions which affect the truth value of the fluent

The conciseness and perspicuity of the solution relies on

- quantification over actions
- the assumption that relatively few actions affect each fluent
- the completeness assumption (for effects)

Moreover, the solution depends on the fact that actions always have deterministic effects.

Limitation: primitive actions

As yet we have no way of handling in the situation calculus complex actions made up of other actions such as

- conditionals: If the car is in the driveway then drive else walk
- iterations: while there is a block on the table, remove one
- nondeterministic choice: pickup up some block and put it on the floor

and others

Would like to *define* such actions in terms of the primitive actions, and inherit their solution to the frame problem

Need a compositional treatment of the frame problem for complex actions

Results in a novel programming language for discrete event simulation and high-level robot control

The Do formula

For each complex action A, it is possible to define a formula of the situation calculus, $Do(A, s, s')$, that says that action A when started in situation s may legally terminate in situation s'.

Primitive actions: $Do(A, s, s') = Poss(A,s) \wedge s'=do(A,s)$

Sequence: $Do([A;B], s, s') = \exists s''.\ Do(A, s, s'') \wedge Do(B, s'', s')$

Conditionals: $Do([\textbf{\textit{if}}\ \phi\ \textbf{\textit{then}}\ A\ \textbf{\textit{else}}\ B], s, s') =$
$\phi(s) \wedge Do(A, s, s') \vee \neg\phi(s) \wedge Do(B, s, s')$

Nondeterministic branch: $Do([A \mid B], s, s') = Do(A, s, s') \vee Do(B, s, s')$

Nondeterministic choice: $Do([\pi x.\ A], s, s') = \exists x.\ Do(A, s, s')$

etc.

Note: programming language constructs with a purely logical situation calculus interpretation

GOLOG

GOLOG (Algol in logic) is a programming language that generalizes conventional imperative programming languages

- the usual imperative constructs + concurrency, nondeterminism, more...
- bottoms out *not* on operations on internal states (assignment statements, pointer updates) but on *primitive actions* in the world (*e.g.* pickup a block)
- what the primitive actions do is user-specified by precondition and successor state axioms

What does it mean to "execute" a GOLOG program?

- find a sequence of primitive actions such that performing them starting in some initial situation s would lead to a situation s' where the formula $Do(A, s, s')$ holds
- give the sequence of actions to a robot for actual execution in the world

Note: to find such a sequence, it will be necessary to reason about the primitive actions

A ; ***if*** Holding(x) ***then*** B ***else*** C

to decide between B and C we need to determine if the fluent *Holding* would be true after doing A

GOLOG example

Primitive actions: pickup*(x)*, putonfloor*(x)*, putontable*(x)*

Fluents: Holding*(x,s)*, OnTable*(x,s)*, OnFloor*(x,s)*

Action preconditions: $Poss($pickup$(x), s) \equiv \forall z.\neg$Holding$(z, s)$

$Poss($putonfloor$(x), s) \equiv$ Holding(x, s)

$Poss($putontable$(x), s) \equiv$ Holding(x, s)

Successor state axioms:

Holding$(x, do(a,s)) \equiv a=$pickup$(x) \vee$
Holding$(x,s) \wedge a\neq$putontable$(x) \wedge a\neq$putonfloor(x)

OnTable$(x, do(a,s)) \equiv a=$putontable$(x) \vee$ OnTable$(x,s) \wedge a\neq$pickup(x)

OnFloor$(x, do(a,s)) \equiv a=$putonfloor$(x) \vee$ OnFloor$(x,s) \wedge a\neq$pickup(x)

Initial situation: $\forall x.\neg$Holding(x, S_0)

OnTable$(x, S_0) \equiv x=$A $\vee x=$B

Complex actions:

proc ClearTable : ***while*** $\exists b.$OnTable(b) ***do*** πb [OnTable$(b)?$; RemoveBlock(b)]

proc RemoveBlock(x) : pickup(x) **;** putonfloor(x)

Running GOLOG

To find a sequence of actions constituting a legal execution of a GOLOG program, we can use Resolution with answer extraction.

For the above example, we have

$$KB \models \exists s.\ Do(\text{ClearTable}, S_0, s)$$

The result of this evaluation yields

$$s = do(\text{putonfloor}(\text{B}), do(\text{pickup}(\text{B}), do(\text{putonfloor}(\text{A}), do(\text{pickup}(\text{A}), S_0))))$$

and so a correct sequence is

$$\langle \text{pickup(A)}, \text{putonfloor(A)}, \text{pickup(B)}, \text{putonfloor(B)} \rangle$$

When what is known about the actions and initial state can be expressed as Horn clauses, the evaluation can be done in Prolog.

The GOLOG interpreter in Prolog has clauses like

```
do(A,S1,do(A,S1)) :- prim_action(A), poss(A,S1).
do(seq(A,B),S1,S2) :- do(A,S1,S3), do(B,S3,S2).
```

This provides a convenient way of controlling a robot at a high level.