PHIL/COGS 3750

Yves Lespérance

Lecture Notes
Reasoning about Action and the Frame Problem

Readings:
See course web site.

Reasoning about Action

A key component of intelligence is the ability to make *plans of action* to achieve one's goals.

This involves reasoning about actions and their effects and preconditions (i.e. when an action can be executed).

Situation Calculus

A popular language for reasoning about action and change in AI is the *Situation Calculus*. It is a dialect of first-order logic.

The constant S_0 is used to represent the initial situation.

The term $do(\alpha, s)$ represents the situation that *results from action* α *being performed* in situation s; e.g. the situation that results from moving block $B1$ onto block $B2$ in the initial situation is represented by $do(move(B1, B2), S_0)$.

Predicates that are affected by action/change take a situation argument; these are called *fluents*.

E.g. can have $\neg On(B1, B2, S_0)$ and $On(B1, B2, do(move(B1, B2), S_0))$.

Special fluent $Poss(\alpha, s)$ represents the fact that action α is *possible/executable* in situation s.

Specifying Action Preconditions

To specify when actions are possible/executable in an application domain, one writes *action precondition axioms*. E.g.

$$Poss(move(x, y), s) \equiv Clear(x, s) \wedge Clear(y, s) \wedge x \neq y.$$

i.e. the robot can move block x onto block y in situation s if and only if they are both clear in s and x and y are distinct.

Specifying Action Effects

We can also specify the effects of actions in a domain by writing *effect axioms*. E.g.

$$On(x, y, do(move(x, y), s))$$

i.e. after moving x onto y, x is on y;

$$On(y, x, s) \wedge x \neq z \supset Clear(x, do(move(y, z), s))$$

i.e. if y is on x and y is moved onto z which is distinct from x, then x becomes clear.

Specifying the Initial Situation

We can also write axioms to specify what is known about the initial situation. E.g.

$$OnTable(B1, S_0), OnTable(B2, S_0),$$
$$OnTable(B3, S_0), OnTable(B4, S_0),$$
$$Clear(B1, S_0), Clear(B2, S_0), Clear(B3, S_0), Clear(B4, S_0).$$

In general, the KB can contain disjunctions and other forms of incomplete knowledge.

Planning

A plan/sequence of actions $a_1, a_2, \ldots, a_n$ *achieves a goal* G in a situation s if the goal G must be true in the situation that results from performing $a_1, a_2, \ldots, a_n$ in situation s and the plan is executable in s, i.e.

$Axioms \models G(do(a_n, \ldots, do(a_1, s)\ldots))$
and
$Axioms \models Executable([a_1, \ldots, a_n], s)$
where
$Executable([a_1, \ldots, a_n], s)) \doteq$
$Poss(a_1, s) \wedge Poss(a_2, do(a_1, s) \wedge \ldots$
$\wedge Poss(a_n, do(a_{n-1}, \ldots, do(a_1, S_0)\ldots)).$

Planning E.g.

Suppose that our goal is to have a 3 blocks tower, i.e.

$$G(s) \doteq \exists x \exists y \exists z\, On(x, y, s) \wedge On(y, z, s) \wedge OnTable(z, s).$$

We can show that the plan $move(B2, B3), move(B1, B2)$ achieves this goal G in the initial situation if we can show that

$Axioms \models Goal(do(move(B1, B2), do(move(B2, B3), S_0)))$
and
$Axioms \models Poss(move(B2, B3), S_0)$
and
$Axioms \models Poss(move(B1, B2), do(move(B2, B3), S_0))).$

The Frame Problem

Even if we have in our KB all the necessary precondition and effect axioms and initial state axioms, this is not enough to show that the plan achieves the goal.

We can show that:

$$Axioms \models On(B2, B3, do(move(B2, B3), S_0))$$

and

$$Axioms \models Poss(move(B2, B3), S_0)$$

But we can't show that

$$Axioms \models OnTable(B1, do(move(B2, B3), S_0))$$

and

$$Axioms \models On(B2, B3, do(move(B1, B2), do(move(B2, B3), S_0))$$

etc.

The reason is that we have not specified what *does not change* when an action is performed!

The Frame Problem (cont.)

One approach is to also provide *frame axioms*. E.g.

$$OnTable(x, s) \wedge y \neq x \supset OnTable(x, do(move(y, z), s)$$

i.e. if x is on the table and y is moved onto z and $y \neq x$, then x must still be on the table afterwards.

But there are *too many* of those. Actions generally affect very few fluents and all others are unaffected.

The Frame Problem (cont.)

The *representational frame problem* is the problem of representing compactly which facts persist when actions are performed.

The *inferential frame problem* is the problem of computing efficiently which facts persist after an action.

STRIPS

There is a simple approach to avoid the frame problem when one has complete information about the state. This is called the *STRIPS* approach.

In STRIPS, one represents a state as the set of all fluents that are true in the state. Actions are specified by providing a list of facts that they add (i.e., positive effects) and a list of facts that they delete (i.e., positive effects). (Preconditions can also be specified.)

One can obtain the state that follows an action by performing the addition and deletions in the current state.

See Section 9.2 of the Levesque textbook for a similar approach.

However, the STRIPS approach breaks down when one does not have complete information; it is not a general solution to the frame problem.

The Ramification Problem

It is straightforward to specify direct effects of action using effect axioms. But actions often have indirect effects/ramifications.

E.g. if a block $B1$ is glued or tied to $B2$ and we move $B1$, then $B2$ will move as well; we could specify this as a state constraint stating that two glued blocks are always next to each other.

The *ramification problem* is how to compactly represent and reason efficiently about the effects of action when there are such state constraints and indirect effects.

The Qualification Problem

Earlier, we assumed that we could specify necessary and sufficient conditions for an action to be possible/executable.

But this is usually not possible. We know of some necessary conditions, e.g.

$$Poss(start(car), s) \supset HasGas(car, s) \wedge BatteryCharged(car, s)$$

But these are not sufficient. There is a huge number of exceptional conditions that could make the action impossible, e.g.

$$EngineStolen(car, s) \supset \neg Poss(start(car), s)$$

$$PotatoInTailpipe(car, s) \supset \neg Poss(start(car), s)$$

State constraints can also lead to such qualifications.

The Qualification Problem (cont.)

Unless we can identify necessary and sufficient conditions and have enough information in the KB to entail them, it is not entailed that an action is in fact executable.

Humans assume that none of these exceptional conditions hold unless they know otherwise. If they know some fact that is relevant to the executability of a plan, they (usually) take it into account and draw the right conclusions.

How can we get a machine to do this?

The *qualification problem* is how to reason effectively about when an action is executable.

Solutions to the Frame Problem

The representational frame problem is considered solved for domains where actions are deterministic (flipping a coin is non-deterministic). A popular solution is Reiter's, which involves taking all the effect axioms involving a given fluent, e.g.

$$\forall x \forall s\, OnTable(x, do(moveToTable(x), s))$$
$$\forall x \forall s\, \neg OnTable(x, do(move(x, y), s))$$

and manipulating them to obtain a successor state axiom, e.g.

$$\forall x \forall s [OnTable(x, do(a, s)) \equiv a = moveToTable(x) \\ \vee\, OnTable(x, s) \wedge \neg \exists y. a = move(x, y)].$$

The STRIPS approach also works when there is complete information.

Current Research Status

The inferential frame problem has been solved in some restricted cases. There is ongoing research in the area. The general case is most likely unsolvable.

There is ongoing research on the ramification problem and qualification problems.

Non-Monotonic Reasoning

These problems, especially the qualification problem, are instances of *non-monotonic reasoning*.

Classical logical entaiment is monotonic. If $KB \models \phi$ and $KB \subseteq KB'$, then $KB' \models \phi$.

But many forms of commonsense reasoning are non-monotonic.

E.g. From *Birds typically fly.*
Tweety is a bird. infer *Tweety flies.*

From *Birds typically fly.*
Penguins are birds.
Penguins typically don't fly.
Tweety is a bird.
Tweety is a penguin. infer *Tweety does not fly.*

Several approaches to non-monotonic reasoning have been developed.