EECS 3401 — AI and Logic Prog. — Lecture 17

Adapted from slides of Brachman & Levesque (2005)

Vitaliy Batusov
vbatusov@cse.yorku.ca

York University

November 16, 2020

- Today: **Actions, Situations, and GOLOG**
- Required reading: R & N Ch.11 (10 in 3-rd ed.)

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.

- Many-sorted logic: has several domains, one for each sort. Each term is interpreted only within its own sort.
- Situation calculus has sorts **action**, **situation**, and **object**

Actions and Situations

Actions: denoted by function terms of sort *action*, e.g.,

- $put(x, y)$ — put thing x on top of thing y
- $walk(location)$ — walk to location *location*
- $pickup(r, x)$ — robot r picks up thing x

Situations: world histories built using specialized symbols S_0 and $do(\cdot, \cdot)$

- S_0 — a constant, always denotes the initial situation
- $do(a, s)$ — a situation that results from doing action a in situation s

Example: $do(put(A, B), do(put(B, C), S_0))$

Fluents

- Already understand: predicates in FOL
- **Fluents**: predicates whose values may **vary from situation to situation**
- Syntactically, a fluent is a predicate whose last argument is of sort *situation*
  Example: $Holding(r, x, s)$ — robot r is holding thing x in situation s
  Can also say things like $\neg Holding(r, x, s) \wedge Holding(r, x, do(pickup(r, x,), s))$
- Note: there is no distinguished "current" situation. A sentence can talk about many different situations, past, present, and future.

Poss

Also:

- A distinguished predicate symbol $Poss(a, s)$ is used to state that action a is *legal to carry out* in situation s.

  $Poss(pickup(r, x), S_0)$ — it is possible for robot r to pick up thing x in the initial situation

Preconditions and effects

- It is necessary to include in a KB not only facts about the initial situation, but also about **world dynamics**
- I.e., a formal account of how and why things which are true (false) in one situation become false (true) in the next
- Action preconditions and action effects

Preconditions and effects

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

$$Poss(pickup(R, X), S) \leftrightarrow \forall Z\,[\neg Holding(R, Z, S)] \\ \wedge \neg Heavy(X) \wedge NextTo(R, X, S)$$

Free variables are assumed to be universally quantified from outside

$$Poss(repair(R, X), S) \leftrightarrow HasGlue(R, S) \wedge Broken(X, S)$$

These are called *precondition axioms*

Preconditions and effects

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

$$Fragile(X) \rightarrow Broken(X, do(drop(R, X), S))$$

$$\neg Broken(X, do(repair(R, X), S))$$

These are called *effect axioms*

The Frame Problem

- Effect axioms only describe what *changes*. To fully describe how the world works, need to specify what fluents are *unaffected* by what actions.

$$Colour(X, C, S) \rightarrow Colour(X, C, do(drop(R, X), S))$$

$$\neg Broken(X, S) \wedge [X \neq Y \vee \neg Fragile(X)] \rightarrow \neg Broken(X, do(drop(R, Y), S))$$

These are called *frame axioms*

Frame Problem

- **Problem!** There will always be a **vast** number of frame axioms.
  An object's colour is unaffected by picking things up, opening a door, calling a friend, turning on a light, weather patterns in China, etc.
- In building a KB, need to include $2 \times |A| \times |F|$ facts about what doesn't change, and then reason efficiently with them

The Frame Problem

- Suppose we have a complete set of effect axioms (non-trivial dynamics, written down by a specialist)
- Can we maybe generate the frame axioms mechanically?
- And, hopefully, in some compact form
- Yes, under some assumptions (later)

The Projection Task

What is the situation calculus good for?

- **Projection task**: *Given a sequence of actions, determine what would be true in the situation that results from performing that sequence*
- More formally: Suppose $R(S)$ is a formula with a free situation variable S. To find out if $R(S)$ would be true after performing actions $\langle a_1, \ldots, a_n \rangle$ in the initial situation, we determine whether or not

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

- Example: using axioms above, it follows that $\neg Broken(b, S)$ would hold after executing the sequence

$$\langle pickup(a), pickup(B), drop(B), repair(B), drop(A) \rangle$$

Legality / Executability

- Projection does not test for whether the sequence of actions is legal (wrt precondition axioms)
- We call a situation *legal* is it is the initial situation or the result of performing an action whose preconditions are satisfied starting in a legal situation
- **Legality task**: task of determining whether a sequence of actions leads to a legal situation
- More formally: To find out if the sequence $\langle a_1, \ldots, a_n \rangle$ can be legally performed in the initial situation, we determine whether or not

$$KB \models Poss(a_i, do(a_{i-1}, \ldots, do(a_1, S_0) \ldots))$$

for all $1 \leq i \leq n$.

Limitations of Situation Calculus

(as presented)

- 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
- 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 internal structure to actions, conditional actions, iterative actions, etc.

A Solution to Frame Problem

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

$$Fragile(X) \rightarrow Broken(X, do(drop(R, X), S))$$
$$NextTo(B, X, S) \rightarrow Broken(X, do(explode(B), S))$$

- Can equivalently rewrite these as

$$\exists R\{A = drop(R, X) \wedge Fragile(X)\}$$
$$\vee \exists B\{A = explode(B) \wedge NextTo(B, X, S)\}$$
$$\rightarrow Broken(X, do(A, S))$$

A Solution to Frame Problem

- Similarly for the negative effect axiom:

$$\neg Broken(X, do(repair(R, X), S))$$

can be rewritten as

$$\exists R\{A = repair(R, X)\} \rightarrow \neg Broken(X, do(A, S))$$

Note how nice the right-hand sides are starting to look. This is called the *normal form for effect axioms*. (One positive NF axiom, one negative NF axiom.)

A Solution to Frame Problem

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

$$P_F(\bar{X}, A, S) \rightarrow F(\bar{X}, do(A, S)) \tag{1}$$

$$N_F(\bar{X}, A, S) \rightarrow \neg F(\bar{X}, do(A, S)) \tag{2}$$

- Next, make a *completeness assumption* regarding these:
  Assume that (1) and (2) characterize ALL the conditions under which an action A changes the value of fluent F
- Formally, this is captured by *explanation closure axioms*:

$$\neg F(\bar{X}, S) \wedge F(\bar{X}, do(A, S)) \rightarrow P_F(\bar{X}, A, S) \tag{3}$$

$$F(\bar{X}, S) \wedge \neg F(\bar{X}, do(A, S)) \rightarrow N_F(\bar{X}, A, S) \tag{4}$$

A Solution to Frame Problem

- In fact, axioms (3) and (4) are, in fact, disguised versions of the frame axioms!

$$\neg F(\bar{X}, S) \wedge \neg P_F(\bar{X}, A, S) \rightarrow \neg F(\bar{X}, do(A, S))$$

$$F(\bar{X}, S) \wedge \neg N_F(\bar{X}, A, S) \rightarrow F(\bar{X}, do(A, S))$$

A Solution to Frame Problem

- Need some additional assumptions:
  - Integrity of effect axioms: can't have $P_F(\bar{X}, A, S)$ and $N_F(\bar{X}, A, S)$ hold at the same time—this must be provable from KB
  - Unique names for actions—some standard axioms

- With these and some effort, it can be shown that, in the models of the KB, the axioms (1)–(4) are logically equivalent to

$$F(\bar{X}, do(A, S)) \leftrightarrow P_F(\bar{X}, A, S) \vee F(\bar{X}, S) \wedge \neg N_F(\bar{X}, A, S)$$

  This is called the **successor state axiom** for F.

Example

Example of a SSA:

$$\begin{aligned} Broken(X, do(A, S)) \leftrightarrow & \exists R\{A = drop(R, X) \wedge Fragile(X)\} \\ & \vee \exists B\{A = explode(B) \wedge NextTo(B, X, S)\} \\ & \vee Broken(X, S) \wedge \neg\exists R\{A = repair(R, X)\} \end{aligned}$$

A simple solution to the frame problem

- This simple solution is due to Raymond Reiter yields the following axioms:
  - one SSA per fluent
  - one precondition axiom per action
  - unique name axioms for actions
- Note: the length of a SSA is roughly proportional to the number of actions which affect the truth value of the fluent
- The conciseness of the solution relies on quantification over actions, the assumption that relatively few actions affect each fluent, and the completeness assumption (also, assumption of determinism)

Limitation: primitive actions

- With the above, we have no way of handling complex actions made up of other actions, such as
  - **conditionals**: If a car is in the driveway, then drive, else walk
  - **iterations**: while there is a block on the table, remove one
  - **non-deterministic choice**: pick up some block and put in on the floor
  - etc.
- 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 simulations and high-level robot control

The *Do* formula

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

$$\begin{aligned}
\text{Primitive action:} \quad & Do(A, s, s') \triangleq Poss(A, s) \wedge s' = do(A, s) \\
\text{Sequence:} \quad & Do([A; B], s, s') \triangleq \exists s'' \left[Do(A, s, s'') \wedge Do(B, s'', s') \right] \\
\text{Conditional:} \quad & Do([\text{if } \phi \text{ then } A \text{ else } B], s, s') \triangleq \\
& \qquad \phi(s) \wedge Do(A, s, s') \vee \neg\phi(s) \wedge Do(B, s, s') \\
\text{Nondet. branch:} \quad & Do([A \mid B], s, s') \triangleq Do(A, s, s') \vee Do(B, s, s') \\
\text{Nondet. choice:} \quad & Do([\pi x.A], s, s') \triangleq \exists x \, Do(A, s, s')
\end{aligned}$$

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

GOLOG

GOLOG: (Al**go**l in **log**ic) is a programming language that generalizes conventional imperative programming languages

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

GOLOG

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!

Consider a program

$$A; [\text{if } Holding(x) \text{ then } B \text{ else } C]$$

Here, to decide between B and C, we need to determine if the fluent $Holding(x)$ would be true *after* executing A

GOLOG example

Primitive actions: $pickup(x)$, $putonfloor(x)$, $putontable(x)$
Fluents: $Holding(x, s)$, $OnTable(x, s)$, $OnFloor(x, s)$

Action preconditions:

$$Poss(pickup(x), s) \leftrightarrow \forall z. \neg Holding(z, s)$$
$$Poss(putonfloor(x), s) \leftrightarrow Holding(x, s)$$
$$Poss(putontable(x), s) \leftrightarrow Holding(x, s)$$

Successor state axioms:

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

Initial situation:

$$\forall x \neg Holding(x, S_0)$$
$$OnTable(x, S_0) \leftrightarrow x = A \vee x = B$$

GOLOG example

Complex actions:

$$\textbf{proc}\ \mathit{ClearTable}:$$
$$\quad \textbf{while}\ \exists b.\mathit{OnTable}(b)\ \textbf{do}\ \pi b[\mathit{OnTable}(b)?; \mathit{RemoveBlock}(b)]$$
$$\textbf{proc}\ \mathit{RemoveBlock}(x): \quad \mathit{pickup}(x); \mathit{putonfloor}(x)$$

Running GOLOG

- To find a sequence of actions constituting a legal execution of a GOLOG program, we can use Resolution with answer extractions
- For the above example, we have

$$KB \models \exists s\ Do(ClearTable, S_0, s)$$

with s determined through unification as

$$s = do(putonfloor(B), do(pickup(B), do(putonfloor(A), do(pickup(A), S_0))))$$

and so a correct sequence is

$$\langle pickup(A), putonfloor(A), pickup(B), putonfloor(B) \rangle$$

Running GOLOG

- 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).
```

  Compare this to the logical definitions of *Do*.
- This provides a way of controlling an agent at a high level

GOLOG Demo

(A Quick Demo)

Planning (again)

- We saw how an agent could figure out what to do given a high-level program or complex action to execute
- Now, consider a related but more general reasoning problem: figure out what to do to make an arbitrary condition true.
  - This is the definition of the **planning problem**
  - The condition to be achieved is called the **goal**
  - The sequence of actions that will make the goal true is called the **plan**
- Recall: different levels of abstraction
- In practice, planning involves anticipating what the world will be like, but also observing the world and replanning as necessary

Planning using Situation Calculus

- Situation calculus can be used to represent what is known about the current state of the world and the available actions
- The planning problem can then be formulated as follows.

> Given a formula $Goal(s)$, find a sequence of actions $\alpha_1, \ldots, \alpha_n$ such that
>
> $$KB \models Goal(do(\langle \alpha_1, \ldots, \alpha_n \rangle, S_0)) \wedge Legal(do(\langle \alpha_1, \ldots, \alpha_n \rangle, S_0))$$
>
> where $do(\langle \alpha_1, \ldots, \alpha_n \rangle, S_0)$ is an abbreviation for $do(\alpha_n, do(\alpha_{n-1}, \ldots, do(\alpha_2, do(\alpha_1, S0)) \ldots))$ and $Legal$ implements the notion of legality from last lecture.

So, given a goal formula, we want a sequence of actions such that (a) the goal formula holds in the situation that results from executing the actions, and (b) it is possible to execute each action in the corresponding situation

Planning by Answer Extraction

- Having formulated planning in this way, we can use Resolution with answer extraction to find a sequence of actions:

$$KB \models \exists s\, (Goal(s) \wedge Legal(s))$$

- We can see how this will work using a simplified version of a previous example:
  - An object is on the table that we would like to have on the floor. Dropping it will put it on the floor, and we can drop it, provided we are holding it. To hold it, we need to pick it up, and we can always to so.

KB

$OnFloor(x, do(drop(x), s))$

$Holding(x, do(pickup(x), s))$

$Holding(x, s) \rightarrow Poss(drop(x), s)$

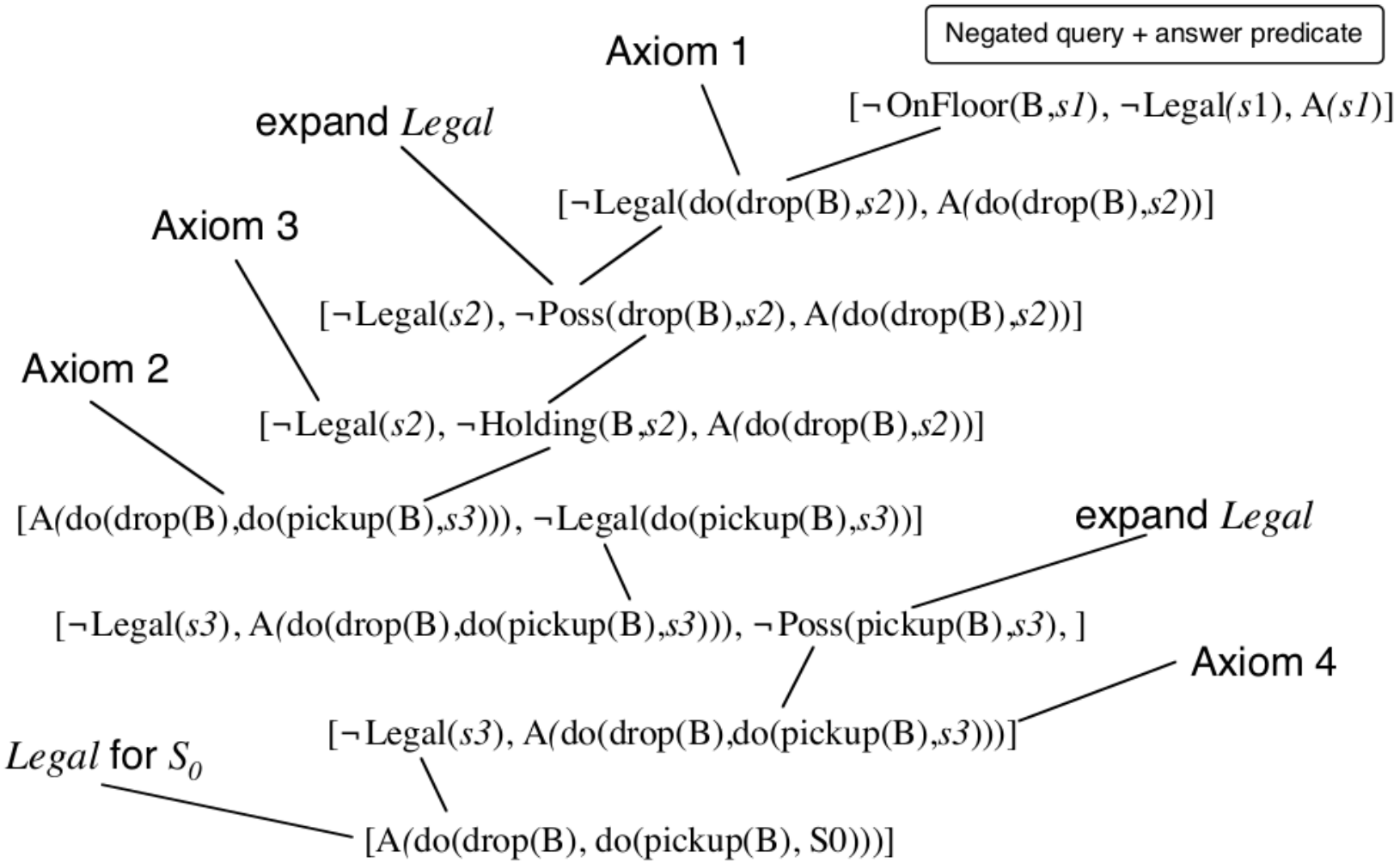
$Poss(pickup(x), s)$

$OnTable(B, S_0)$

Goal

$OnFloor(B, s)$

Deriving a Plan



Using Prolog

Because all the required facts here can be expressed as Horn clauses, we can use Prolog directly to synthesize a plan:

```
onfloor(X,do(drop(X),S)).
holding(X,do(pickup(X),S)).
poss(drop(X),S) :- holding(X,S).
poss(pickup(X),S).
ontable(b,s0).
legal(s0).
legal(do(A,S)) :- poss(A,S), legal(S).
```

With the prolog goal `?- onfloor(b,S), legal(S).` 1 we get the solution `S = do(drop(b),do(pickup(b),s0))` .
(Demo)

Planning in GOLOG

- Consider: ***while*** $\neg Goal$ ***do*** $\pi a.a$
- Planning problem in GOLOG
- Planning problems are typically hard
- Too hard for blind search
- Too hard for Resolution alone
- Not to mention that entailment in FOL is undecidable

- In search: heuristics
- In FOL: ***while*** $\neg Goal$ ***do*** $\pi a.[Acceptable(a)?; a]$,
  where $Acceptable(a)$ describes domain-dependent guidance.