EECS 3401 — AI and Logic Prog. — Lecture 18

Adapted from slides of Brachman & Levesque (2005)

Vitaliy Batusov
vbatusov@cse.yorku.ca

York University

November 23, 2020

- Today: **GOLOG, Planning, Intro to Uncertain Reasoning**
- Required reading: Russell & Norvig Ch.13 and Ch. 14.1, 14.2

Recap: 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'

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

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

Recap: GOLOG

To *execute* a GOLOG program A is to 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

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:

$$\begin{aligned} Holding(x, do(a, s)) &\leftrightarrow a = pickup(x) \vee \\ &\qquad Holding(x, s) \wedge a \neq putontable(x) \wedge a \neq putonfloor(x) \\ OnTable(x, do(a, s)) &\leftrightarrow a = putontable(x) \vee OnTable(x, s) \wedge a \neq pickup(x) \\ OnFloor(x, do(a, s)) &\leftrightarrow a = putonfloor(x) \vee OnFloor(x, s) \wedge a \neq pickup(x) \end{aligned}$$

Initial situation:

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

GOLOG example

Complex actions:

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

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

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.

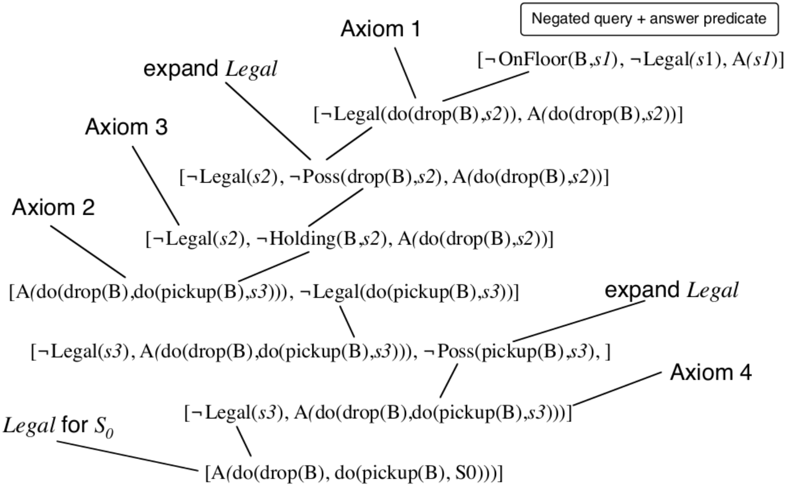
KB

$$OnFloor(x, do(drop(x), s))$$
$$Holding(x, do(pickup(x), s))$$
$$Holding(x, s) \rightarrow Poss(drop(x), s)$$
$$Poss(pickup(x), s)$$
$$OnTable(B, S_0)$$

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).` we get the solution `S = do(drop(b),do(pickup(b),s0))`.
(Demo)

Planning in GOLOG

- Consider: $\textbf{\textit{while}}\ \neg Goal\ \textbf{\textit{do}}\ \pi a.a$
- 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: $\textbf{\textit{while}}\ \neg Goal\ \textbf{\textit{do}}\ \pi a.[Acceptable(a)?; a]$,
  where $Acceptable(a)$ describes domain-dependent guidance.

New Topic
Vagueness, Uncertainty, and Degrees of Belief

- Ordinary common-sense knowledge quickly moves away from categorical statements like *a P is always a Q*
- Many ways to come up with less categorical information
  - things are usually (almost never; occasionally; seldomly; rarely; almost always) a certain way
  - judgments about how good an example something is ("barely rich", "not very tall", etc.)
  - imprecision of sensors
  - reliability of sources of information
  - strength/confidence/trust in generic information or deductive rules
- With information like this, conclusions will not "follow" in the usual sense

Weakening a Universal

There are at least three ways a universal can be made to be less categorical

$$\forall x\, P(x)$$

1. *Strength of quantifier*
   "95% of birds can fly"
   statistical interpretation, probabilistic sentences
2. *Degree of belief* in the whole sentence
   "80% confidence in this fact"
   uncertain knowledge, subjective probability
3. *Applicability of predicate / degree of membership*
   "fairly tall"
   flexible membershit, **vague predicates**

Objective probability

Statistical (frequency) view of sentences

- **Objective**: does not depend on who is assessing the probability
- Always applied to collections (as opposed to singleton random events)
- Can use probabilities to correspond to English words like "rarely", "likely", "usually"
- Generalized quantifiers
  Compare: *for most* x, $Q(x)$ vs. *for all* x, $Q(x)$

Basic postulates

- Real numbers between 0 and 1 representing frequency of an event in a large-enough random sample
- $0 =$ "never happens", $1 =$ "always happens"
- Start with set U of all possible occurrences. An event a is any subset of U. A **probability measure** is any function $\boldsymbol{Pr}$ from events to $[0, 1]$ satisfying
  - $\boldsymbol{Pr}(U) = 1$
  - If $a_1, \ldots, a_n$ are disjoint events, then $\boldsymbol{Pr}(\cup_i a_i) = \sum_i \boldsymbol{Pr}(a_i)$

Basic postulates

- Conditioning: the probability of one event may depend on its interaction with others

$$\boldsymbol{Pr}(a \mid b) = \frac{\boldsymbol{Pr}(a \cap b)}{\boldsymbol{Pr}(b)}$$

- Conditional independence: event a is judged *independent* of event b conditional on background knowledge s if knowing that b happened does not affect the probability of a

$$\boldsymbol{Pr}(a \mid s) = \boldsymbol{Pr}(a \mid b, s)$$

Some consequences

- Conjunction:

$$\begin{aligned} \boldsymbol{Pr}(ab) &= \boldsymbol{Pr}(a \mid b) \cdot \boldsymbol{Pr}(b) && \text{(in general)} \\ &= \boldsymbol{Pr}(a) \cdot \boldsymbol{Pr}(b) && \text{(conditionally indep.)} \end{aligned}$$

- Negation:

$$\boldsymbol{Pr}(\neg s) = 1 - \boldsymbol{Pr}(s)$$
$$\boldsymbol{Pr}(\neg s \mid d) = 1 - \boldsymbol{Pr}(s \mid d)$$

Some consequences

- If $b_1, b_2, \ldots, b_n$ are pairwise disjoint and exhaust all possibilities, then

$$\boldsymbol{Pr}(a) = \sum_i \boldsymbol{Pr}(ab_i) = \sum_i \boldsymbol{Pr}(a \mid b_i) \cdot \boldsymbol{Pr}(b_i)$$
$$\boldsymbol{Pr}(a \mid c) = \sum_i \boldsymbol{Pr}(ab_i \mid c)$$

- Bayes' rule

$$\boldsymbol{Pr}(a \mid b) = \frac{\boldsymbol{Pr}(a) \cdot \boldsymbol{Pr}(b \mid a)}{\boldsymbol{Pr}(b)}$$

If a is a disease and b a symptom, it is usually easier to estimate numbers on the right-hand side