15.

Planning

Planning

So far, in looking at actions, we have considered how an agent could figure out what to do given a high-level program or complex action to execute.

Now, we consider a related but more general reasoning problem: figure out what to do to make an arbitrary condition true. This is called planning.

- the condition to be achieved is called the goal
- the sequence of actions that will make the goal true is called the plan

Plans can be at differing levels of detail, depending on how we formalize the actions involved

- "do errands" vs. "get in car at 1:32 PM, put key in ignition, turn key clockwise, change gears,…"

In practice, planning involves anticipating what the world will be like, but also observing the world and replanning as necessary...

Using the situation calculus

The 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 $\boldsymbol{a}$ such that
>
> $$KB \models Goal(do(\boldsymbol{a}, S_0)) \wedge Legal(do(\boldsymbol{a}, S_0))$$
>
> where $do(\langle a_1,...,a_n\rangle, S_0)$ is an abbreviation for
>
> $$do(a_n, do(a_{n-1}, ..., do(a_2, do(a_1, S_0)) ...))$$
>
> and where $Legal(\langle a_1,...,a_n\rangle, S_0)$ is an abbreviation for
>
> $$Poss(a_1, S_0) \wedge Poss(a_2, do(a_1, S_0)) \wedge \ ... \wedge Poss(a_n, do(\langle a_1,...,a_{n-1}\rangle, S_0))$$

So: given a goal formula, we want a sequence of actions such that

- the goal formula holds in the situation that results from executing the actions, and
- it is possible to execute each action in the appropriate situation

Planning by answer extraction

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

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

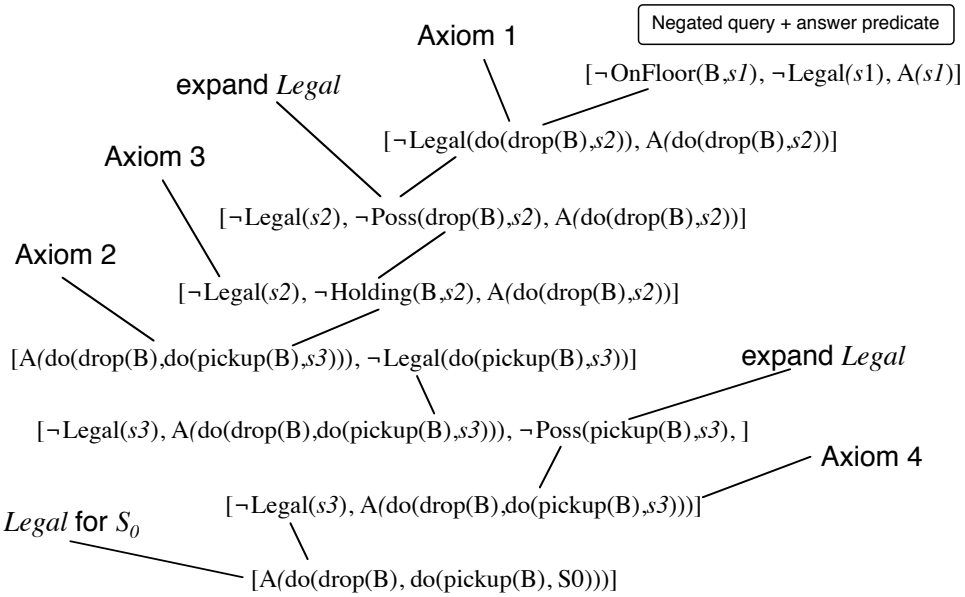
We can see how this will work using a simplified version of a previous example:

> 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 do so.

- Effects: $\text{OnFloor}(x, do(\text{drop}(x), s))$
  $\text{Holding}(x, do(\text{pickup}(x), s))$
  Note: ignoring frame problem
- Preconds: $\text{Holding}(x, s) \supset Poss(\text{drop}(x), s)$
  $Poss(\text{pickup}(x), s)$
- Initial state: $\text{OnTable}(\text{B}, S_0)$
- The goal: $\text{OnFloor}(\text{B}, s)$

KB

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))`

But planning problems are rarely this easy!

Full Resolution theorem-proving can be problematic for a complex set of axioms dealing with actions and situations explicitly...