

1 Administrivia

The data file for HW2 Question 3 is now available.

Reminder 1: Homework 2 is due this Thurs.

Reminder 2: Midterm exam is next Thurs. It will cover material up through this Thurs, but not next week's material.

Reminder 3: Grad students – your semester project proposal is due in a couple of weeks. You should be thinking about it now. You're welcome to come talk to me about it (in fact, I *strongly recommend* that you do so.)

2 Planning as Theorem Proving; STRIPS

- Reading for this lecture: Sec 7.4.
- So we've talked a bit about how to prove theorems in an automated way. But what do you *use* that ability for?
- Well, the obvious thing is to use it to prove mathematical theorems. There's a large body of literature on doing that, and people have had some substantial successes with it.
- But you can *also* use it to reason about the world and how the world works.
- In particular, you can use it to talk about *planning* – that is, automatically figuring out how to act in the world.
- The planning task is: given a description of how the world works, a set of actions for manipulating the world, and a description of a goal state, figure out what sequence of actions will take you from the current state to the desired future state.
- Sounds a lot like search, doesn't it?
- Well, it is, but we're usually interested in worlds that are best described with predicate calc, so we want to carry out our search in FOPL terms rather than in more primitive (propositional) representations.
- Example: the canonical example of this class of planning is the "blocks world" problem. You read a bit about Brook's contempt of this problem in today's reading, but it's still a useful example problem to look at.
- In this problem, you have a (usually entirely hypothetical) table with a set of colored (and possibly shaped) blocks sitting on it.
- There's also a robot manipulator that can pick up, move, and set down blocks.
- The usual scenario is that you start with some set of blocks piled in some configuration on the table, and you want to get to a different configuration of piled blocks.

- Example: we have a table with 3 blocks: a red, green, and yellow block. Initially, the red block is on the table, the green block is on top of it, and the yellow block is on the table. (We'll assume for the moment that the color uniquely identifies a block, so we can name each block only by its color.) The robot gripper is currently empty.
- First, we describe the situation in FOPL.
- We need a set of predicates that describe relationships in the world: `on(X, Y)`, `ontable(X)`, `clear(X)`, `gripping(X)`, `gripping()` (indicating that the robot isn't holding anything), and `clear(X)` (indicating that block X doesn't have anything sitting on it.)
- Given those descriptions, the current situation is described as:

$$\text{ontable}(\text{red}) \wedge \text{ontable}(\text{yellow}) \wedge \text{on}(\text{green}, \text{red}) \wedge \text{gripping}() \\ \wedge \text{clear}(\text{green}) \wedge \text{clear}(\text{yellow})$$

- In planning terminology, these predicates are called **fluents**. Fluents are things that can change from time step to time step.
- Next, we need to specify our goal. Let's say that we're trying to build a traffic light, so we want red on top of yellow on top of green. We write this as:

$$\text{ontable}(\text{green}) \wedge \text{on}(\text{yellow}, \text{green}) \wedge \text{on}(\text{red}, \text{yellow})$$

- We're leaving out a lot of other description – like what (if anything) the gripper is holding and whether the red block is clear. The idea is that other factors are irrelevant – only the things we explicitly enumerate in the goal description matter. (So, for example, if our goal is *Rich* \wedge *Famous*, then the conclusion *Rich* \wedge *Famous* \wedge *Miserable* does meet our conditions, even if it's not quite what we want. As always, the computer gives us what we *ask* for, not what we *want*...)
- Now we need a description of how the world works. Basically, we're looking for a set of predicates that express what actions are available to us and what they do. In the blocks world, there are four commands that we'll worry about: `pickup(X)` and `putdown(X)` which take a block from and return it to the table, and `stack(X, Y)` and `unstack(X, Y)` which put blocks on top of one another or take them off.
- Our description of our actions have to take into account how the world works and make sure that things remain consistent. I.e., we want to be sure that we can't pick something up from the bottom of the stack, etc.
- We have to introduce a new operator \leftarrow . $Q \leftarrow P$ means “remove the expression P from the knowledge base and replace it with expression Q ”. This isn't, strictly speaking, a FOPL operation, though you could finesse (i.e., hack) it into being one. It would be painful, though, and it's simpler to think about just removing and replacing.
- We'll start with a set of three axioms that are independent of the actions – these just make sure that the world remains in a consistent state.

1. $\forall X(\text{clear}(X) \leftarrow \neg \exists Y(\text{on}(Y, X)))$
2. $\forall Y(\forall X(\text{on}(Y, X) \leftarrow \text{ontable}(Y)))$
3. $\forall Y(\text{gripping}() \leftrightarrow \neg \text{gripping}(Y))$

Note that these can also be seen as descriptions of *how to achieve some state of affairs*. I.e., if we want to get to the state of affairs in which the gripper is empty ($\text{gripping}()$), we have to ensure that it's not holding any block. This seems obvious to us, but by default, the computer doesn't know of any connection between the predicate $\text{gripping}()$ and the predicate $\text{gripping}(\text{green})$. It might be possible for $\text{gripping}()$ and $\text{gripping}(\text{green})$ to hold at the same time – we have to explicitly tell it that that can't happen.

Next, we have to describe how the actions work. These rules basically say how actions link different descriptions (states) of the world together.

4. $\forall X(\text{pickup}(X) \rightarrow (\text{gripping}(X) \leftarrow \text{gripping}() \wedge \text{clear}(X) \wedge \text{ontable}(X)))$
5. $\forall X(\text{putdown}(X) \rightarrow ((\text{gripping}() \wedge \text{ontable}(X) \wedge \text{clear}(X)) \leftarrow \text{gripping}(X)))$
- 6.

$$\forall X(\forall Y(\text{stack}(X, Y) \rightarrow ((\text{on}(X, Y) \wedge \text{gripping}() \wedge \text{clear}(X)) \leftarrow (\text{clear}(Y) \wedge \text{gripping}(X))))))$$

7.

$$\forall X(\forall Y(\text{unstack}(X, Y) \rightarrow ((\text{clear}(Y) \wedge \text{gripping}(X)) \leftarrow (\text{on}(X, Y) \wedge \text{clear}(X) \wedge \text{gripping}()))))$$

Finally, we need a set of axioms that say, in effect, “these rules don't have any side effects”. Basically, we need to be able to express that the world remains largely unchanged – that things remain the same from step to step unless explicitly acted upon. These are called **frame axioms** and they are used to address the **frame problem** – i.e., the question of what does and doesn't change in the world when you take an action. (Note that we're assuming that our robot is the *only* one acting on the world at any time and that we don't have to worry about other agents, randomness, natural catastrophes, etc.) As an aside, the term *frame* comes from “frame of reference” in physics – the “background” against which you measure things.

8. $\forall X(\forall Y(\forall Z(\text{unstack}(Y, Z) \rightarrow (\text{ontable}(X) \leftarrow \text{ontable}(X))))))$
9. $\forall X(\forall Y(\forall Z(\text{stack}(Y, Z) \rightarrow (\text{ontable}(X) \leftarrow \text{ontable}(X))))))$

- The last two say, effectively, that `stack` and `unstack` don't mess with things other than their arguments. Clearly, we *also* have to write down a bunch of frame axioms that express what (doesn't) happen when we do `pickup` or `putdown` or that `stack` doesn't affect already clear-ness or stacked-ness for things other than its arguments. Obviously, this quickly becomes a pain in the butt.
- Assuming that you *can* write down all the necessary action and frame axioms, then you can stuff the whole system into your resolution theorem prover and it will (in principle) find a sequence of actions that will transform your input state into your goal state.

- One problem with this is that theorem proving is still hard – the search process that it’s doing is still nasty. This will be a problem no matter *how* you do the theorem proving, though.
- A bigger issue is that it’s just a real pain in the butt to have to specify all the frame axioms. Note that if you have A actions and F predicates that describe the world, then you need $O(AF)$ frame axioms to specify what doesn’t change. That’s pretty nasty for even reasonably large worlds. And note that *all* of those then become clauses in the knowledge base on which resolution could work – i.e., they increase the branching factor pretty frighteningly.
- Another problem is that this representation just doesn’t fit how we think about the world very well – it’s very hard for humans to figure out how to write down the effects of actions. And it’s even *harder* for humans to make sure that they have all of the frame axioms correct. Finally, every time you add or change an action rule or predicate, you have to update the frame axioms – very tedious and error-prone.
- People pretty quickly gave up on trying to do planning with brute-force theorem proving. While it can work (and has been shown to), it’s just not convenient, and often not practical.
- As a response, people quickly developed a different representation – **STRIPS**.
- STRIPS (the STanford Reasearch Institute Planning System) makes a major simplifying assumption: the **closed world assumption**. This says, in essence, that nothing changes unless some rule explicitly says that it changes.
- Thus, you avoid having to explicitly enumerate all of the frame axioms. This saves a lot of time and effort and lets the search process be much more efficient, but it *is* an assumption.
- STRIPS still describes the world in terms of a small set of fluents, but rather than using a FOPL description of actions, it uses a special action representation.
- Each action is made up of three elements:
 1. A set of **preconditions** – a set of logical descriptions of what must be true in the world in order for this action to be relevant.
 2. An **add list** – a set of new fluents that hold after the action is taken.
 3. A **delete list** – the set of old fluents that are deleted after the action is taken.
- It can be shown that rules of this form have exactly the same representational power (and logical correctness) as the FOPL planning representation, but these are *much* easier to work with.