CompSci 366

Introduction to Planning

What's Happening

- 4 Lectures Covering:
  - Classical Plan Representation
  - Progression Planning
  - Regression Planning

Outline

- Why plan?
- Current Status & Future of Planning
- Modeling a Domain
- Plans and Planning

Why Plan?

- We have a problem!
  - We have a goal that isn't currently achieved.

Why Plan?

- We have a problem!
  - We have a goal that isn't currently achieved.
- We don't readily know how to solve it!
  - Need to think about how to solve it.

Why Plan?

- We have a problem!
  - We have a goal that isn't currently achieved.
- We don't readily know how to solve it!
  - Need to think about how to solve it.
- What we want is a plan!
  - Don't simply want to know if it is achievable, want a plan that will achieve it.

How Do We Plan?

- We usually do mental simulations of scenarios.
- Not just one scenario, first one rarely succeeds.
- But a search through a space of different scenarios for one that achieves our goal.
- These scenarios describe our *problem space*

Problem Space Hypothesis

- Allen Newell proposed the *Problem Space Hypothesis*: namely that all human problem-solving can be described in terms of:
  - Situations
  - Operators that have preconditions and effects.
  - Search control knowledge.
- Planners are certainly built this way.

Should We Always Plan?

- In general, planning is very computationally expensive!!!!

Should We Always Plan?

- In general, planning is very computationally expensive!!!!
- Even in very simplified models of our domains!!

Should We Always Plan?

- In general, planning is very computationally expensive!!!!
- Even in very simplified models of our domains!!
- In everyday life, caching is usually better than planning.

When Should An Agent Plan?

- When it has a problem.

When Should An Agent Plan?

- When it has a problem.
- That it doesn’t already have a cached solution for.

When Should An Agent Plan?

- When it has a problem.
- That it doesn't already have a cached solution for.
- But where it has a domain model which will allow it to explore possible solution scenarios.

Our First Glimpse of Classical Planning

- Will describe an approach to :
  - Domain modelling *(this lecture)*
  - Search space structure *(next lecture)*
  - Search space traversal *(next lecture)*

Domain Modelling

- Classical assumptions
- Problem representation
- Situation representation
- Goal Representation
- Action representation
- The simulation process
- Plan representation

Classical Assumptions

- Finite propositional domains.
- Omniscience.
- Actions are completely deterministic.
- No exogenous events.

Classical Assumptions

- All actions take unit time to occur.
- Only qualitative resources.
- All actions occur sequentially.

Domain Modelling

- Classical assumptions
- Problem representation
- Situation representation
- Goal Representation
- Action representation
- The simulation process
- Plan representation

Problem Representation

- Problem is described by:
  - A complete description of the initial situation.
  - A description of the desired goals.

Domain Modelling

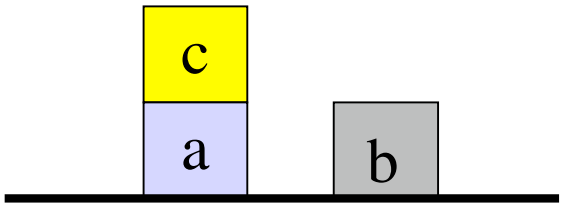
- Classical assumptions
- Problem representation
- ➔ Situation representation
- Goal Representation
- Action representation
- The simulation process
- Plan representation

Situation Representation

- Omniscience => must be able to determine the truth/falsity of all propositions for a given situation.
- Generally, situations cannot be represented by vectors.
- Situations are represented by propositional expressions.

Situation Representation cont'd

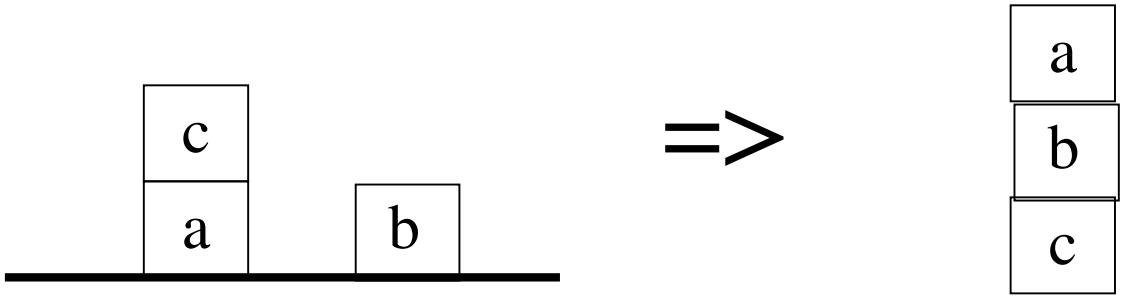
- Given a situation, how do we represent it?



Situation Representation cont'd

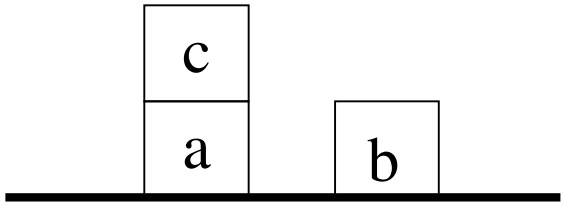
- Need to do some domain ontology engineering
  - What aspects are important?
  - What do we want to capture?
- In general, very difficult to get right.

Sample Problem



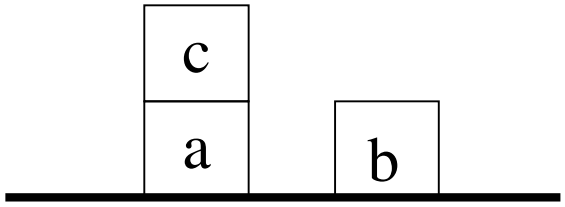
- What's important and needs to be represented?
- Assume we only need the following predicates:
  - **on(Block1, Thing1)**
  - **clear(Block)**

Situation Representation cont'd



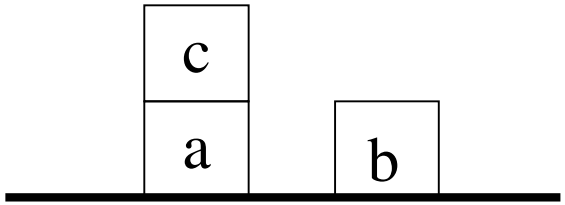
- Given our ontology, how would we describe this situation?
- Need to know everything about situation.

Situation Representation cont'd



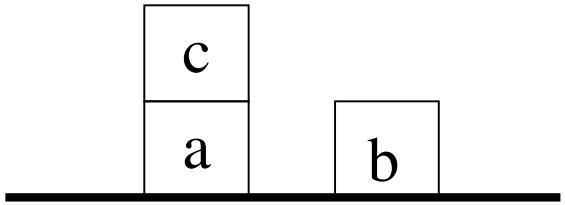
- What about : **on(c,a), on(a,table), clear(c), on(b, table), clear(b), ~on(c,b), ~on(c,table), on(c,a) ^ on(a,table), on(c,a) v on(a,b), …**

Situation Representation cont'd



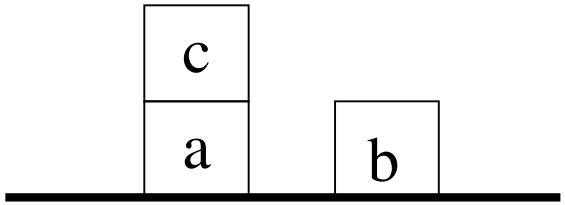
- Negated literals: an infinite number of things that aren't true in a given situations.
- Logical expressions: an infinite number of true expressions can be built up from logical combinations of literals.

Situation Representation cont'd



- Negated literals: use closed world assumption (CWA) [similar to negation by failure in Prolog].
- Logical expressions: use deduction to evaluate.

Situation Representation cont'd



- Situation = **{on(c,a), on(a,table), clear(c), on(b, table), clear(b)}**
- Want to be able to infer **~on(c,b)**, etc., and **on(c,a) ^ on(a,table)**, etc., from this representation.

Situation Representation cont'd

- Some things about the world we can change or some we can't, represented by values of predicates changing .
- The predicates whose values can change from state to state are called ***fluents***.
- The domain operators tell us which predicates are fluents, since we assume no other agents and no exogeneous events.
- For example, the location of the warehouse keeper in sokoban is a fluent as are the location of the tiles in the 8-puzzle.

Situation Representation cont'd

- There are also some predicates whose values will not change for a problem, i.e., are ***static***, though they may differ from problem to problem.
- For example, in sokoban the location of the walls is static during a problem and in 8-puzzle which positions are adjacent to which other positions is static.
- Static predicates never appear in the effects of an op.
- Also, static predicates can be associated with the problem rather than being associated with each state.

Situation Representation cont'd

- Storing the static information separately from the fluents can enable a large savings in space and time.
- However, this means that there must be a record of which predicates are fluent and which are static, so the program knows whether to look at the state or the problem to determine its value.

Situation Representation cont'd

- The values of some predicates can be derived from the values other predicates.
- For example, in blocks world the value of the clear(Block) predicate can be derived from the value of the on(X, Block) predicate, i.e., if there is nothing on top of Block then it is clear.
- Such predicates are called ***derived*** predicates as opposed to the ***primitive*** predicates (predicates whose values can not be derived from other the values of other predicates).

Situation Representation cont'd

- If the planner can actually use a definition of the derived predicate to calculate its value, then we can save space by not storing derived predicates (or we can save time by caching their computed values).
- If the planner calculates the values of derived predicates then we need to differentiate between derived and primitive predicates because the values of primitive fluents will be stored with each state while the definitions of derived predicates may be defined as part of the domain information and their values may either be computed on demand or cached.
- For example, we could define:

  clear(Block) :- block(Block), once(not(on(_, Block))).
  clear(table).

Domain Modelling

- Classical assumptions
- Problem representation
- Situation representation
- ➔ Goal Representation
- Action representation
- The simulation process
- Plan representation

Goal Representation

- In situations, the fact that **on(b,c)** is *not true* is presented by not having it in the list of true literals.

- How would we represent the goal of wanting **on(b,c)** to be *false*?

Goal Representation

- For states, there are only 2 options a literal is either true or it is false.
- For goals, there are 3 options, a literal is desired to be true, desired to be false, or we don't care.
- With goals we need to distinguish between what we want to be false and what we don't care about.

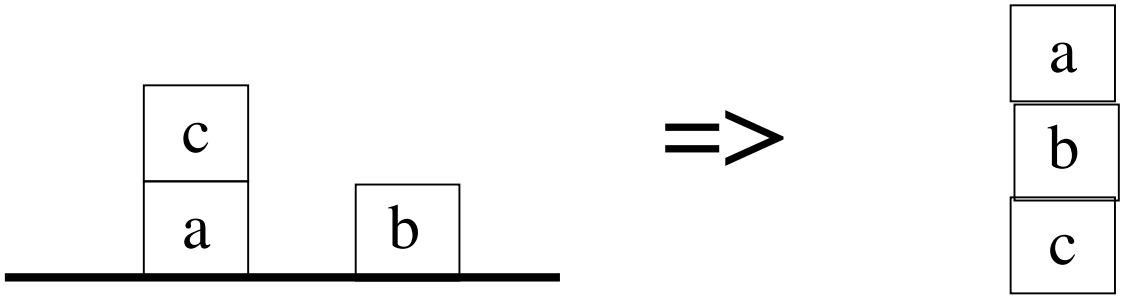
Goal Representation

- How would we represent the goal of wanting **on(b,c)** to be *false* versus not caring about it?

- For us, we represent it as **not(on(b,c))**.

- Literals we don't care about are not mentioned in the goal description.

Problem Representation Revisited



- Initial situation = **{on(c,a), on(a,table), clear(c), on(b, table), clear(b)}**
- Goal situation = **{on(a,b), on(b,c)}**, note haven't said what, if anything, is on top of **a** nor what **c** is on top of, it's a "don't care".

Domain Modelling

- Classical assumptions
- Problem representation
- Situation representation
- Goal Representation
- Action representation
- The simulation process
- Plan representation

Action Representation

- Propositionally based.
- Use action schema rather than concrete actions (i.e., parameterised actions).
- Need to describe:
  - When action is legal.
  - What things remain the same when the action is executed.
  - What things change when the action is executed.
  - Perhaps, when the action makes sense.

Action Representation cont'd

- Action schema represents actions that will take place in our simulations and reflect what happens in our domain.
- Action representation includes:
  - Action schema name.
  - Action schema parameters.
  - Action preconditions.
  - Action effects.

Action Representation cont'd

- The parameters indicate what objects are relevant to executing this action.
- The preconditions describe when the action can be executed.
- The effects describe what happens when the action is executed.

Action Representation cont'd

- Describing the effects of executing an action is a bit like describing a state.
- We would like the description to be short but also complete.
- The problem of establishing exactly what changes and what does not change when an action is executed is called the *frame* problem.

Action Representation cont'd

- For example, just like a state description could list all the false literals associated with the state, the effects could list all the things that remain untouched by executing the action.
- Similar to the use of the CWA to decrease the number of things that need to be specified for a state, we can use another assumption to decrease the length of the effects list.
- We will assume that the effects only describe what changes, and anything not mentioned remains the same.

Action Representation cont'd

- Example action schema:
  - Move block from location to location.
  - op*(*move, [Block, FromLoc, ToLoc],
    [on(Block,FromLoc), clear(Block), clear(ToLoc)],
    [not(on(Block,FromLoc)), clear(FromLoc),
    not(clear(ToLoc)), on(Block, ToLoc)] *)*
- What's wrong with this schema?

Action Representation cont'd

- What happens when **ToLoc** is the table?

Action Representation cont'd

- What happens when **ToLoc** is the table?
  - Can only move blocks to the table at certain times.
  - Moving a block to the table makes the table no longer clear (whatever that means, what does "clear" mean anyway?).
  - Then can't move anything to it until it's made clear again.
- How could we fix this?

Action Representation cont'd

- One way: extend domain language by introducing new action for moving blocks to table: **newStack(Block, FromLoc)**
- Also, check that **move**'s **ToLoc** is a block: add block type check to preconditions and add block type info to situation descriptions.

Action Representation cont'd

- **move** action schema revisited:
  - op*(*move, [Block, FromLoc, ToBlock],
    [on(Block,FromLoc), clear(Block), block(ToBlock),
    clear(ToBlock)],
    [not(on(Block,FromLoc)), clear(FromLoc),
    not(clear(ToBlock)), on(Block, ToBlock)] *)*
- What else is wrong with this schema?

Action Representation cont'd

- What happens if **Block = ToBlock**?
  - **move**'s preconditions are satisfied.
  - But end up with a block being on top of itself.
- How do we fix this?

Action Representation cont'd

- One way would be to add the constraint that **Block** and **ToBlock** can't be the same block to **move**'s preconditions: [on(Block,FromLoc), ..., clear(ToBlock)] => [on(Block,FromLoc), …, clear(ToBlock), *neq(Block,ToBlock)*]

Action Representation cont'd

- Note that *neq(Block,ToBlock)* is a different kind of test from **clear(ToBlock)**.
- The latter tests a situation, while the former tests a planner choice.
- The latter is called an *object-level test* and the former is called a *meta-level test*.

Action Representation cont'd

- Is our description of the **move** action good enough now? How can we tell?

Action Representation cont'd

- Is our description of the **move** action good enough now? How can we tell?
- *It's good enough if the plans our planner creates using these descriptions usually (almost always, …) succeed when we execute them in the "real world".*
- How might it fail?

Action Representation cont'd

- It could fail if our preconditions don't capture all the relevant tests.
- In general there are an infinite number of preconditions for the represented action to adequately model the real world action.
- This representation problem is known as the *qualification* problem.

Action Representation cont'd

- It could also fail if our effects don't capture all the relevant results.
- In general there are an infinite number of effects for the represented action to adequately model the real world action.
- This representation problem is known as the *ramification* problem.

Domain Modelling

- Classical assumptions
- Problem representation
- Situation representation
- Goal Representation
- Action representation
- The simulation process
- Plan representation

Simulation Process

- How do we use the action descriptions to simulate the effect of executing that action in a given situation?
  - The parameters must be instantiated.
  - Simulation checks that the action's instantiated preconditions are satisfied by the situation.
  - Positive effect literals are added to the situation and negative effect literals are removed from it.

Domain Modelling

- Classical assumptions
- Problem representation
- Situation representation
- Goal Representation
- Action representation
- The simulation process
- Plan representation

Plans & Planning

- Given a problem description with an initial situation description and a goal description, find a plan that transforms initial situation into one that satisfies the goal description.
- How are we going to represent a plan?
  - As a sequence (i.e., list) of steps.
  - A *step* is an instantiated action schema that is part of a plan. Note: there may be many steps that have the same instantiated action schema.

Plans & Planning cont'd

- Note: given a plan and an initial situation, we can simulate the effect of executing each step of the plan upon the resulting situations.

Plans & Planning cont'd

- How do we find an adequate plan?
- One way is the following:
  - Transform the problem description into an initially "empty" plan.
  - Add actions into the "partial" plan until it represents a solution to the problem.
    - A solution to a problem is a plan that transforms the initial state into a state that satisfies the *goal test*.

Summary

- Planning is how an intelligent agent figures out how to achieve its goals.
- Planning is becoming quite important as we attempt to build evermore competent agents.
- Newell's Problem Space Hypothesis: problem-solving can be described as states, operators, and search control knowledge.

Summary cont'd

- A problem is an initial situation and a goal description.
- If we use CWA then a situation is a set of positive grounded object-level predicates.
- Predicates can be fluent/static, derived/primitive, object-level/meta-level.
- A goal description is a set of possibly positive and negative, object- and meta-level, primitive and derived, and fluents and static-terms.
- An operator's effects can only be a set of positive/negative, object-level, primitive fluents.

Summary cont'd

- A plan is a sequence of steps.
- A step is a domain action.
- A domain action has a name, set of parameters, precondition, and effects.

Summary cont'd

- Creating the predicates and the action descriptions is part of domain engineering.

- Describing domain actions has the following problems:
  - Qualification Problem
  - Ramification Problem
  - Frame Problem

Summary cont'd

- We can create pseudo-steps that represent the initial situation and the goal description.
- With these pseudo-steps, we can create empty plans.
- Planning becomes a plan refinement process.
- We will look at one such process next time.