272SM: Introduction to Artificial Intelligence

Automated Planning

[Some slides were created by Ahmad Bassiouny, https://www.slideshare.net/ahmad1957/planning-presentation-769459.]

Planning problem

- Find a **sequence of actions** that achieves a given **goal** when executed from a given **initial world state**. That is, **given**
  - a set of operator descriptions (defining the possible primitive actions by the agent),
  - an initial state description, and
  - a goal state description or predicate,

  compute a plan, which is
  - a sequence of operator instances, such that executing them in the initial state will change the world to a state satisfying the goal-state description.
- Goals are usually specified as a conjunction of goals to be achieved

Planning vs. problem solving

- Planning and problem solving methods can often solve the same sorts of problems
- Planning is more powerful because of the representations and methods used
- States, goals, and actions are decomposed into sets of sentences (usually in first-order logic)
- Subgoals can be planned independently, reducing the complexity of the planning problem
- Search often proceeds through a much smaller *plan space* rather than *state space* (though there are also state-space planners) by considering only relevant actions

Typical assumptions

- Atomic time: Each action is indivisible
- No concurrent actions are allowed (though actions do not need to be ordered with respect to each other in the plan)
- Deterministic actions: The result of actions are completely determined there is no uncertainty in their effects
- Agent is the sole cause of change in the world
- Agent has complete knowledge of the state of the world
- Closed World Assumption: everything known to be true in the world is included in the state description. Anything not listed is false.
- This constitutes the so-called **classic planning environment**

Blocks world

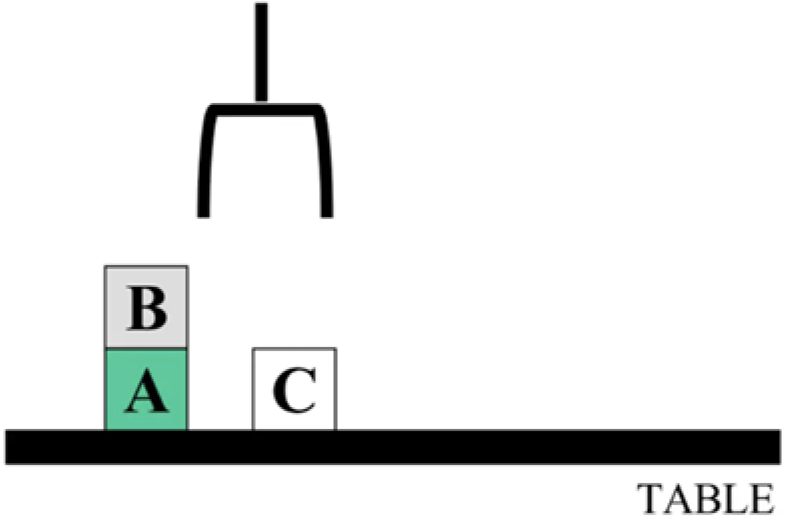
The **blocks world** is a micro-world that consists of a table, a set of blocks and a robot hand.

Some domain constraints:

- Only one block can be on another block
- Any number of blocks can be on the table
- The hand can only hold one block

Typical representation:

ontable(a)
ontable(c)
on(b,a)
handempty
clear(b)
clear(C)



Basic representations for planning

- Classic approach first used in the **STRIPS** planner circa 1970
- States represented as a conjunction of ground literals
  - at(Home) ^ ~ have(Milk) ^ ~have(bananas) ...
- Goals are conjunctions of literals, but may have variables which are assumed to be existentially quantified
  - at(?×) ^ have(Milk) ^ have(bananas) ...
- Do not need to fully specify state
  - Non-specified either don't-care or assumed false
  - Represent many cases in small storage
  - Often only represent changes in state rather than entire situation
- Unlike theorem prover, not seeking whether the goal is true, but is there a sequence of actions to attain it

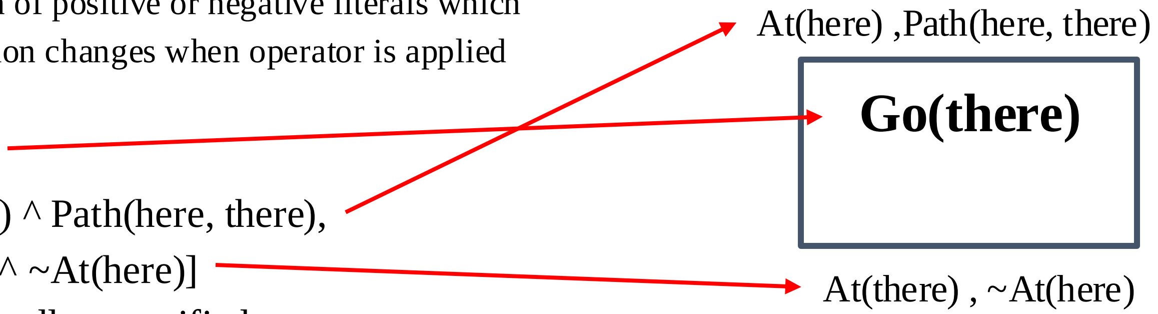
Operator/action representation

- Operators contain three components:
  - **Action description**
  - **Precondition -** conjunction of positive literals
  - **Effect -** conjunction of positive or negative literals which describe how situation changes when operator is applied
- Example:

  Op[Action: Go(there),

  Precond: At(here) ^ Path(here, there),

  Effect: At(there) ^ ~At(here)]
- All variables are universally quantified
- Situation variables are implicit
  - preconditions must be true in the state immediately before operator is applied; effects are true immediately after



Blocks world operators

- Here are the classic basic operations for the blocks world:
    - stack(X,Y): put block X on block Y
    - unstack(X, Y): remove block X from block Y
    - pickup(X): pickup block X
    - putdown(X): put block X on the table
- Each will be represented by
    - a list of preconditions
    - a list of new facts to be added (add-effects)
    - a list of facts to be removed (delete-effects)
    - optionally, a set of (simple) variable constraints
- For example:

```
preconditions(stack(X,Y), [holding(X),clear(Y)])
deletes(stack(X, Y), [holding(X),clear(Y)]).
adds(stack(X, Y), [handempty,on(X, Y), clear(X)])
constraints(stack(X, Y), [XI==Y,Y\==table,X\==table])
```

Blocks world operators II

```
operator(stack(X,Y),
    Precond [holding(X),clear(Y)],
    Add [handempty,on(X, Y), clear(X)],
    Delete [holding(X),clear(Y)].
    Constr [X\==Y, Y\==table, X\==table]).
```

```
operator(unstack(X, Y),
    [on(X,Y), clear(X), handempty],
    [holding(X),clear(Y)],
    [handempty,clear(X),on(X,Y)],
    [X\==Y, Y\==table, X\==table]).
```

```
operator(pickup(X),
    [ontable(X), clear(X), handempty],
    [holding(X)],
    [ontable(X),clear(X),handempty],
    [X\==table]).
```

```
operator(putdown(X),
    [holding(X],
    [ontable(X),handempty,clear(X)],
    [holding(X)],
    [X\==table]).
```

STRIPS planning

- STRIPS maintains two additional data structures:
  - **State List** - all currently true predicates.
  - **Goal Stack** - a push down stack of goals to be solved, with current goal on top of stack.
- If current goal is not satisfied by present state, examine add lists of operators, and push operator and preconditions list on stack. (Subgoals)
- When a current goal is satisfied, POP it from stack.
- When an operator is on top stack, record the application of that operator on the plan sequence and use the operator's add and delete lists to update the current state.

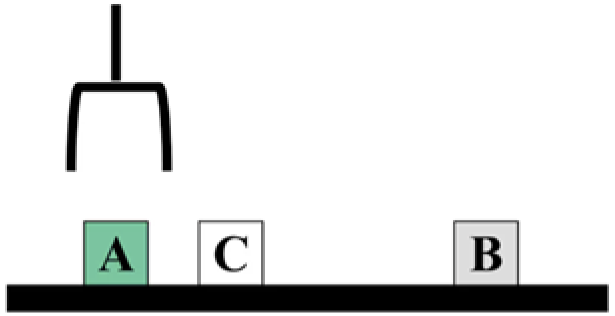
Typical BW planning problem

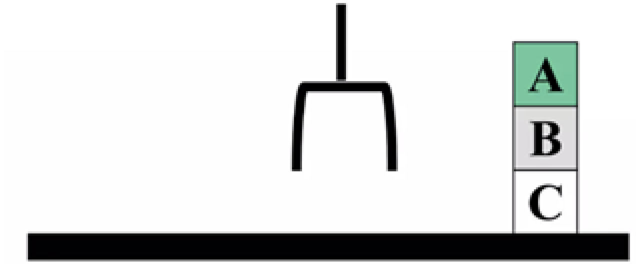
Initial state:

- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:

- on(b,c)
- on(a,b)
- ontable(c)





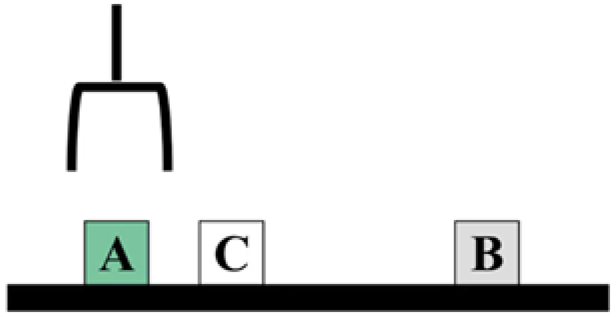
A plan:

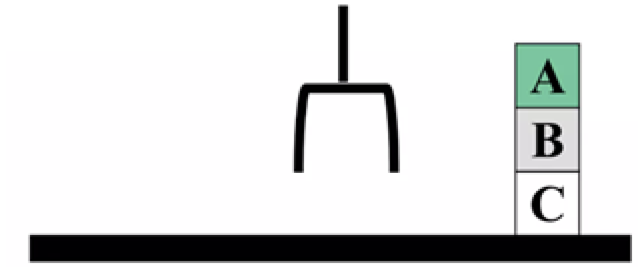
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

Typical BW planning problem

Initial state:
- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:
- on(a,b)
- on(b,c)
- ontable(c)

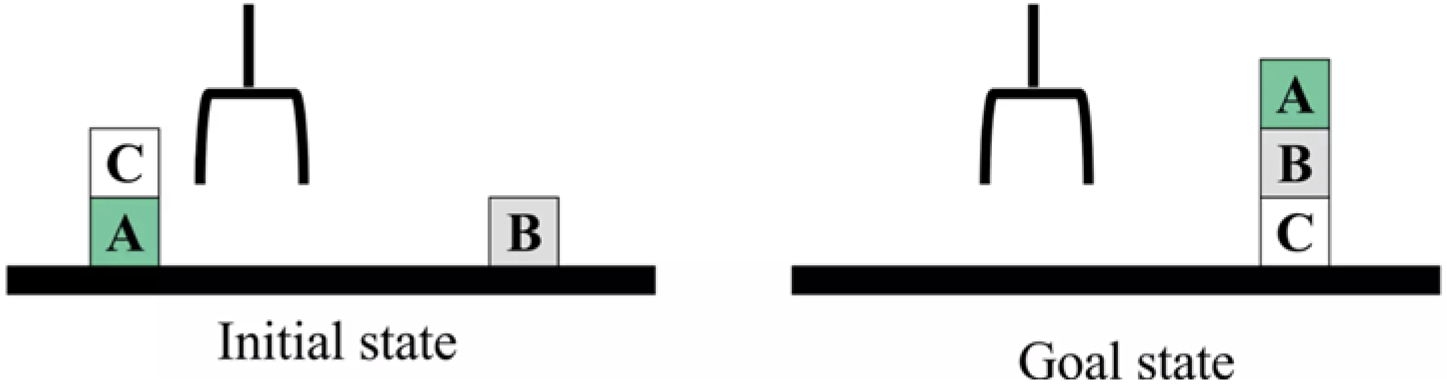




A plan:
- pickup(a)
- stack(a,b)
- unstack(a,b)
- putdown(a)
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

Goal interaction

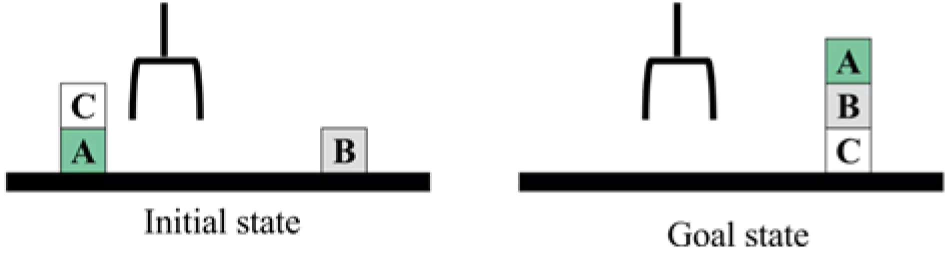
- Simple planning algorithms assume that the goals to be achieved are independent
  - Each can be solved separately and then the solutions concatenated
- This planning problem, called the "Sussman Anomaly," is the classic example of the goal interaction problem:
  - Solving on(A,B) first (by doing unstack(C,A), stack(A,B) will be undone when solving the second goal on(B,C) (by doing unstack(A,B), stack(B,C)).
  - Solving on(B,C) first will be undone when solving on(A,B)
- Classic STRIPS could not handle this, although minor modifications can get it to do simple cases



Sussman Anomaly

```
Achieve on(a,b) via stack(a b) with preconds: [holding(a),clear(b)]
|Achieve holding(a) via pickup(a) with preconds: [ontable(a),clear(a),handempty]
||Achieve clear(a) via unstack(_1584,a) with preconds:
[on(_1584,a),clear(_1584), handempty]
||Applying unstack(c,a)
||Achieve handempty via putdown(_2691) with preconds: [holding(_2691)]
||Applying putdown(c)
[Applying pickup(a)
Applying stack(a,b)
Achieve on(b,c) via stack(b,c) with preconds: [holding(b),clear(c)]
|Achieve holding(b) via pickup(b) with preconds: [ontable(b),clear(b), handempty]
|Achieve clear(b) via unstack(_5625,b) with preconds:
[on(_5625,b),clear(_5625),handempty]
||Applying unstack(a,b)
||Achieve handempty via putdown(_6648) with preconds: [holding(_6648)]
||Applying putdown(a)
|Applying pickup(b)
Applying stack(b,c)
Achieve on(a,b) via stack(a,b) with preconds: [holding(a),clear(b)]
|Achieve holding(a) via pickup(a) with preconds: [ontable(a),clear(a),handempty]
|Applying pickup(a)
Applying stack(a,b)
```

```
From
[clear(b), clear(c),ontable(a), ontable(b),on
(c,a),handempty]
  To [on(a,b),on(b,c),ontable(c)]
  Do:
    unstack(c,a)
    putdown(c)
    pickup(a)
    stack(a,b)
    unstack(a,b)
    putdown(a)
    pickup(b)
    stack(b,c)
    pickup(a)
    stack(a,b)
```



State-space planning

- We initially have a space of situations (where you are, what you have, etc.)
- The plan is a solution found by "searching" through the situations to get to the goal
- A **progression planner** searches forward from initial state to goal state
- A **regression planner** searches backward from the goal
    - This works if operators have enough information to go both ways
    - Ideally this leads to reduced branching -you are only considering things that are relevant to the goal

Plan-space planning

- An alternative is to **search through the space of *plans***, rather than situations.
- Start from a **partial plan** which is expanded and refined until a complete plan that solves the problem is generated.
- **Refinement operators** add constraints to the partial plan and modification operators for other changes.
- We can still use STRIPS-style operators:
  Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)
  Op(ACTION: RightSock, EFFECT: RightSockOn)
  Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)
  Op(ACTION: LeftSock, EFFECT: leftSockOn)

could result in a partial plan of

[RightShoe, LeftShoe]

Partial-order planning

- A **linear planner** builds a plan as a **totally ordered sequence** of plan steps
- A **non-linear planner (aka partial-order planner)** builds up a plan as a set of steps with some temporal constraints
    - constraints of the form SI<S2 if step S1 must comes before S2.
- One **refines** a partially ordered plan (POP) by either:
    - **adding a new plan step**, or
    - **adding a new constraint** to the steps already in the plan.
- A POP can be **linearized** (converted to a totally ordered plan)

by topological sorting

Least commitment

- Non-linear planners embody the principle of **least commitment**
  - only choose actions, orderings, and variable bindings that are absolutely necessary, leaving other decisions till later
  - avoids early commitment to decisions that don't really matter
- A linear planner always chooses to add a plan step in a particular place in the sequence
- A non-linear planner chooses to add a step and possibly some temporal constraints

Non-linear plan

- A non-linear plan consists of

  (1) A set of **steps** $\{S_1, S_2, S_3, S_4\ldots\}$

  Each step has an operator description, preconditions and post-conditions

  (2) A set of **causal links** $\{ \ldots (S_i,C,S_j) \ldots\}$

  Meaning a purpose of step S, is to achieve precondition C of step S_j

  (3) A set of **ordering constraints** $\{ \ldots S_i<S_j \ldots \}$

  if step S_i must come before step S_j
- A non-linear plan is **complete** iff
  - Every step mentioned in (2) and (3) is in (1)
  - If S_j has prerequisite C, then there exists a causal link in (2) of the form (S_i,C,S_j) for some S_i
  - If (S_i, C,S_j) is in (2) and step S_k is in (1), and S_k threatens (S_i, C,S_j) (makes C false), then (3) contains either $S_k <S_i$ or $S_k>S_j$

http://www.cs.ubc.ca/labs/lci/CIspace/

Tools for learning

CIspace Computational Intelligence

Applets to illustrate graph search, CSP, decision trees, belief networks, neural networks, deductive reasoning, planning and robot control. You can run them via the web or download them to your own computer.

Planning

CIspace

The planning applet uses the STRIPS representation to demonstrate the STRIPS, regression, and partial order planners.