CS 4649/7649
Robot Intelligence: Planning

Classical Planning Summary

Sungmoon Joo

School of Interactive Computing
College of Computing
Georgia Institute of Technology

*Slides based in part on Dr. Dana S. Nau and Dr. Mike Stilman's lecture slides

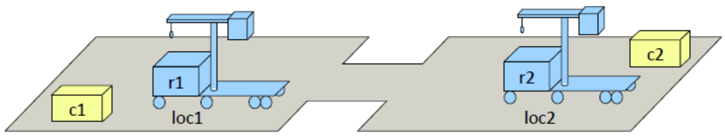
Final Project

- Voice annotated slide show will be uploaded
- Questions? - schedule, deliverables

Classical Planning

Classical Representation

- DWR example:
  - **Constant symbols**: c1, c2, loc1, loc2, r1, r2
  - **Variable symbols** *x, y, …*
  - **Predicates**:
    - adjacent(*l*,*m*) - location *l* is adjacent to location *m*
    - loc(*r*,*l*) - robot *r* is at location *l*
    - pos(*c*,*l*), pos(*c*,*r*) - container *c* is at location *l* or on robot
    - loaded(*r*) - there is a container on robot *r*
- Some terminology
  - **Atom***: predicate symbol and its arguments
  - **Ground**: contains no variable symbols – e.g. loc(r,l) vs loc(r1, loc1)

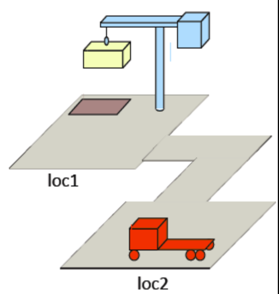


*In logic**, literal** is an atomic formula (atom) or its negation

Classical Planning

Abstraction

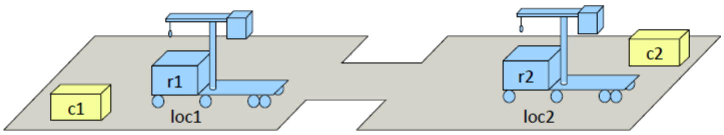
- Real world is absurdly complex, need to approximate
  - Only represent what the planner needs to reason about
- **State transition system** $\Sigma = (S,A,E,\gamma)$
  - S = {abstract states}
    - e.g., states might include a robot's location, but not its position and orientation
  - A = {abstract actions}
    - e.g., "move robot from loc2 to loc1" may need complex lower-level implementation
  - E = {abstract exogenous events}
    - Not under the agent's control
  - γ = state transition function
    - Gives the next state, or possible next states, after an action or event
    - $\gamma: S \times (A \cup E) \rightarrow S$ or $\gamma: S \times (A \cup E) \rightarrow 2^S$



Classical Planning

State

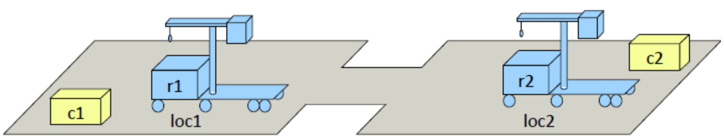
- Use ground atoms to represent both rigid and varying properties of a system Σ
- To represent a state σ of Σ
  - *s* = {all ground atoms of *L* that are true in σ}
  - e.g., *s*1 = {pos(c1,loc1), loc(r1,loc1), pos(c2,loc2), loc(r2,loc2), adjacent(loc1,loc2), adjacent(loc2,loc1)}
- *S* = {all sets of ground atoms of *L*}
  - Call each $s \in S$ a **state**
- *S* may contain some "states" that don't actually represent states of Σ
  - e.g., {pos(c1,loc1), pos(c1,loc2)}
  - Not a big problem if we represent things correctly



Classical Planning

State

- Number of possible states is finite
  - Suppose there are *c* constant symbols
  - *p* predicate symbols, each with *k* arguments
  - Then:
    - » Number of possible ground atoms is pc^k
    - » Number of possible states is 2^{pc^k}



Classical Planning

Operator vs Action

- Operator

$o(x_1,\ldots,x_k)$ — *Name/Head*(Parameter list)

Precond: $p_1, p_2, \ldots, p_h$ — Precond: List of literal, p_i, that must be true for o to be applicable

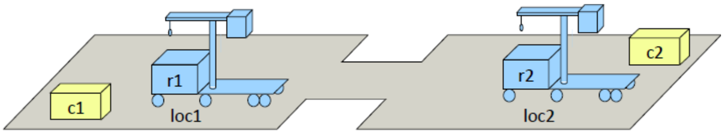
Effects: $e_1, e_2, \ldots, e_h$ — Effects: List of literal, e_i that operator will change

- Action: a ground instance of an operator

take(r,l,c)
Precond: loc(r,l), pos(c,l), ¬loaded(r)
Effects: pos(c,r), ¬pos(c,l),loaded(r)

⇨

take(r1,loc1,c1)
Precond: loc(r1,loc1), pos(c1,loc1), ¬loaded(r1)
Effects: pos(c1,r1), ¬pos(c1,loc1), loaded(r1)

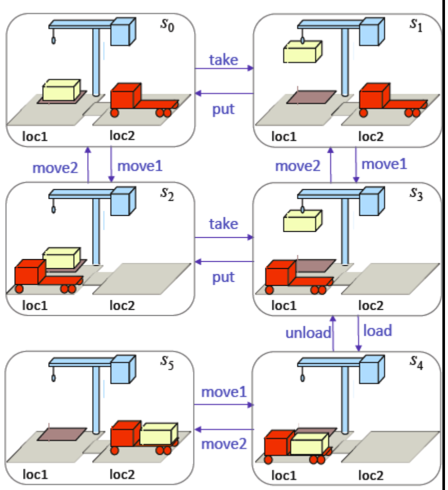


Classical Planning

State Transition

- $\Sigma = (S,A,E,\gamma)$
  S = {states}
  A = {actions}
  E = {exogenous events}
  γ = state-transition func.

- Example:
  S = {s0, …, s5}
  A = {move1, move2, put, take, load, unload}
  E = {}
  - so write $\Sigma = (S,A,\gamma)$
  $\gamma: S \times A \rightarrow S$
  - see the arrows

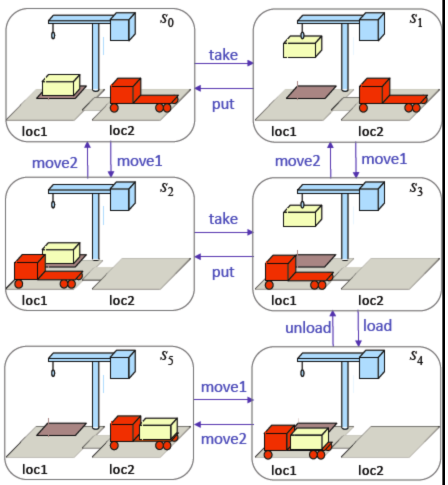


Dock Worker Robots (DWR) example

Classical Planning

Planning Problem

- Description of Σ
- Initial state or set of states
- Objective
  - Goal state, set of goal states, set of tasks, "trajectory" of states, objective function, …
- Example
  - Initial state = s_0
  - Goal state = s_5

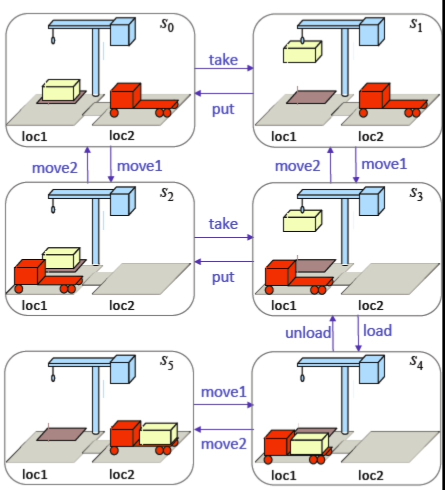


Dock Worker Robots (DWR) example

Classical Planning

Planning Problem

- Description of Σ ← **Domain**
- Initial state or set of states
- Objective
  - Goal state, set of goal states, set of tasks, "trajectory" of states, objective function, …
- Example
  - Initial state = s_0
  - Goal state = s_5

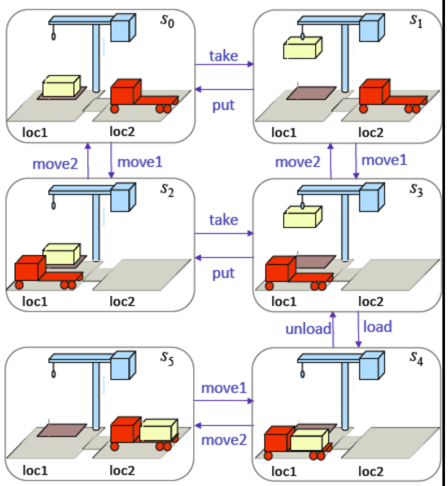


Dock Worker Robots (DWR) example

Classical Planning

Planning Problem

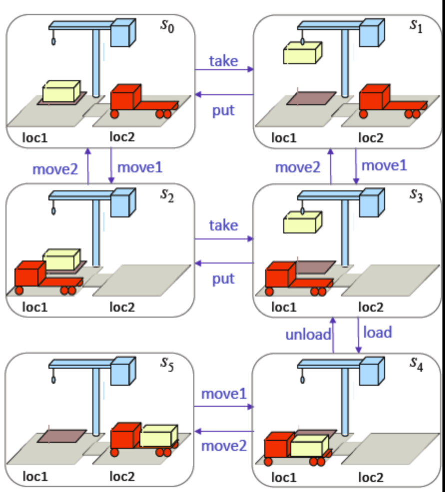
- Description of Σ ← **Domain**
- Initial state or set of states
- Objective
  - Goal state, set of goal states, set of tasks, "trajectory" of states, objective function, …
- Example
  - Initial state = s_0
  - Goal state = s_5



Dock Worker Robots (DWR) example

Classical Planning

- **Classical plan**: a sequence of actions ⟨take, move1, load, move2⟩
- **Policy**: partial function from *S* into *A*
  i.e. State → Action
  e.g.
  {(s_0, take), (s_1, move1), (s_3, load), (s_4, move2)}
- Both, if executed starting at s_0, produce s_3



Dock Worker Robots (DWR) example

Classical Planning

Assumptions

- A0: Finite system: finitely many states, actions, events
- A1: Fully observable: the controller always Σ's current state
- A2: Deterministic: each action has only one outcome
- A3: Static (no exogenous events): no changes but the controller's actions
- A4: Attainment goals: a set of goal states S_g
- A5: Sequential plans: a plan is a linearly ordered sequence of actions (a_1, a_2, … a_n)
- A6: Implicit time: no time durations; linear sequence of instantaneous states
- A7: Off-line planning: planner doesn't know the execution status

Classical Planning

- Classical planning requires all eight restrictive assumptions
  - Offline generation of action sequences for a deterministic, static, finite system, with complete knowledge, attainment goals, and implicit time

- Reduces to the following problem:
  - Given a planning problem $P = (\Sigma, s_0, S_g)$
  - Find a sequence of actions (a_1, a_2, … a_n) that produces a sequence of state transitions (s_1, s_2, …, s_n) such that s_n is in S_g.

- This is just path-searching* in a graph
  - Nodes = states
  - Edges = actions

Situation Calculus*

To represent and reason about dynamical worlds

To represent 'change', 'state (implicitly time)' is introduced.

- **Fluents = Aspects of the world that change**

$$Contains(Suitcase, Laptop, \mathbf{S0})$$
$$Working(Robot, \mathbf{S7})$$

- **Actions** are *reified* functions of constants. (They can be treated as constants themselves)

$$Put(Laptop, Suitcase) \quad Open(Car) \quad Lock(Car)$$

- The **do function**: $do(\alpha, \sigma_0) \rightarrow \sigma_1$

$$\alpha = action$$
$$\sigma = state$$

***Modern version is different from the original for clarity. Calculus = study about 'change'.**

Frame Problem

Effect Axioms (Positive + Negative) describe how a world changes by an action

$$In(Robot, Hall, s) \wedge Open(rm, s) \Rightarrow In(Robot, rm, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$
$$In(Robot, Hall, s) \wedge Open(rm, s) \Rightarrow \neg In(Robot, Hall, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$

For each unchanged fluent we add:

Frame axioms

$$Open(Office, s) \Rightarrow Open(Office, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$
$$\neg Open(Office, s) \Rightarrow \neg Open(Office, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$

$$Color(Sky, Blue, s) \Rightarrow Color(Sky, Blue, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$
$$\neg Color(Sky, Blue, s) \Rightarrow \neg Color(Sky, Blue, \mathbf{do}(\mathbf{enter}(\mathbf{rm}), \mathbf{s}))$$

How many in total? (for n distinct fluents and m distinct actions)

$2nm$ (Not exponential – but often not practical)

Explicitly specify that all conditions not affected by actions are not changed while executing that action

STRIPS

- Represent actions (operators) as three parts:
  - PC: set of preconditions
  - D: Delete List
  - A: Add List

Constants:

$A, B, C, Table$
$IsBlock(A), IsBlock(B)...$

Ground Literals:

$On(B, A)$ $On(C, Table)$
$Clear(B)$ $Clear(Table)$

Actions: $move(x, y, z)$

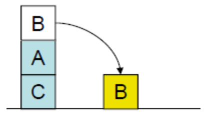
PC: $On(x, y)$, $Clear(x)$, $Clear(z)$

D: $On(x, y)$, $Clear(z)$

A: $On(x, z)$, $Clear(y)$, $Clear(Table)$

Domain Axioms:

$$On(y, x) \wedge On(z, x) \wedge (x \neq Table) \Rightarrow (y = z)$$



$move(B, A, Table)$

S0

Delete:

$On(B, A)$
$Clear(Table)$

Unchanged

$On(A, C)$
$On(C, Table)$
$Clear(B)$

S1

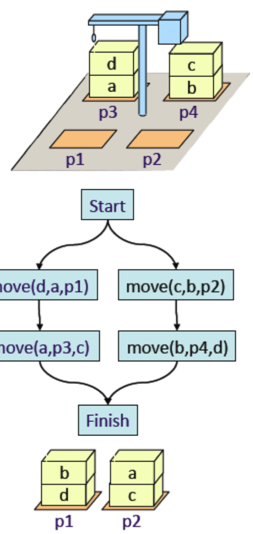
Add:

$On(B, Table)$
$Clear(A)$
$Clear(Table)$

$On(A, C)$
$On(C, Table)$
$Clear(B)$

Plan-Space Planning

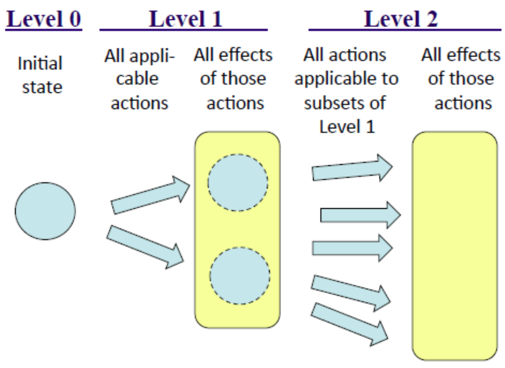
- Decompose sets of goals into the individual goals
- Plan for them separately
  - Bookkeeping info to detect and resolve interactions
- Produce a partially ordered plan that retains as much flexibility as possible
- The Mars rovers used a temporal planning extension of this



Planning Graph

- Idea:
  - First, solve a relaxed problem
    - Each "level" contains all effects of all applicable actions
    - Even though the effects may contradict each other
  - Next, do a state-space search within the planning

- Example
  Graphplan, IPP, CGP, DGP, LGP, SGP,TGP,…



Search

- Uninformed
  - DFS, BFS, IDS

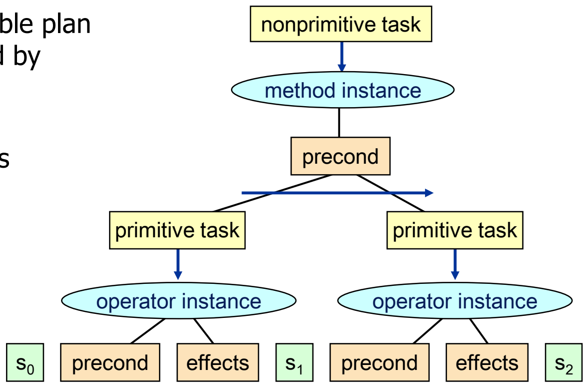
- Informed
  - Best-First-Search (cost-to-go only)
  - A* (cost-to-go+cost-paid)

- Relaxed Planning Graph Heuristic (RPGH)
  - FF(regression search on RPGH), HSP (H_0~H_2)

HTN Planning: Domain, Problem

- STN planning domain: operators, methods
- STN planning problem: domain, initial state, initial task network

- Solution: any executable plan that can be generated by recursively applying
  - methods to non-primitive tasks
  - operators to primitive tasks

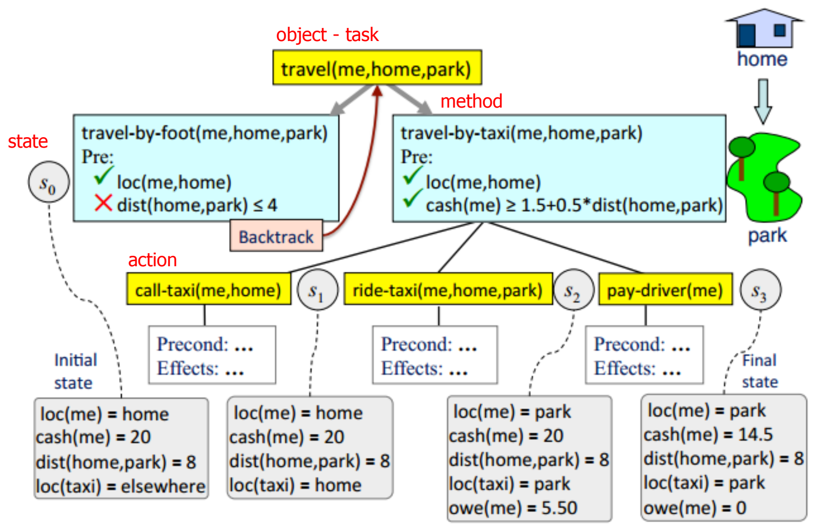


Comparison to Classical Planning

- Like :
  - Each state of the world is represented by a set of atoms.
  - Each action corresponds to a deterministic state transition.
  - Terms, literals, operators, actions, plans have same meaning as classical planning.

- Different:
  - Objective is to perform a set of tasks, not to achieve a set of goals
  - Added tasks, methods, task networks
  - Tasks decompose into subtasks
    - Constraints
    - Task orders
  - Backtrack if necessary

Comparison to Classical Planning



Comparison to Classical Planning

- Advantages
  - Express things that can't be expressed in classical planning
  - Specify(encode) standard ways of solving problems (recipte)

    → Otherwise, the planner have to derive recipes repeatedly from 'first principle' every time it solves a problem

    → Can speed up by orders of magnitude (exponential → polynomial)

- Disadvantages
  - Writing/Debugging an HTN domain model can be cumbersome/complicated

    → try HTN if

    (i) it is important to achieve high performance

    (ii) you need more expressive power than classical planners can provide

Complexity of Planning

Definitions

- **P**
  - If there's an algorithm to solve a problem that runs in polynomial **time** (i.e. can be expressed by some polynomial function of the size of the input)
- **NP** (Non-deterministic Polynomial)
  - If there's an algorithm to solve a problem for which it is not known that it runs in polynomial time
  - It means there is not necessarily a polynomial-time way to find a solution, but once you have a solution it only takes polynomial time to verify that it is correct
  - "non-determinism" refers to the outcome of the algorithm
- **NP-Complete**
  - There is a set of problems in NP for which if there's a polynomial solution to one there will be a polynomial solution to all the set

Complexity of Planning

Definitions

- **PSPACE**
  - If a problem can be solved by an algorithm that uses an amount of **space** polynomial in the size of its input
  - It is known that $P \subset$ PSPACE and $NP \subset$ PSPACE,
  - But, not whether $P \neq$ PSPACE

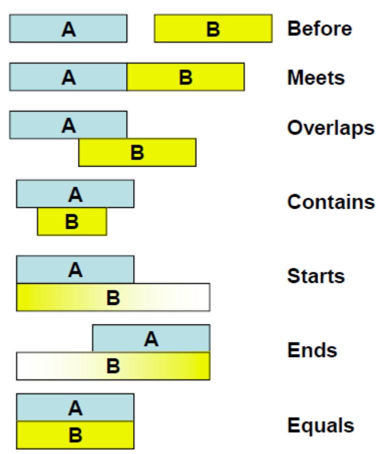
- Given a classical planning problem A, does it have a solution?
  - PSPACE-complete (much harder than NP-complete)
- Given a classical planning problem A and an integer *k*, is there a solution of length *k* or less?
  - PSPACE-complete

Topics not covered

Temporal Planning

If we need an explicit representation of time

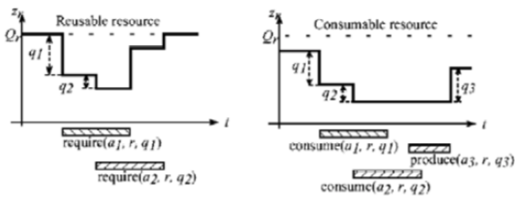
- Actions Have Duration
- Timed Conditions and Effects



Scheduling

- Given:
  – Actions to Perform
  – Set of Resources to Use
  – Constraints on Time

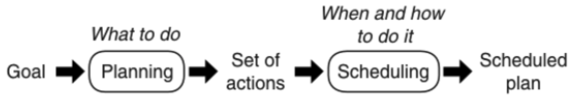
- Goal:
  – Allocate time and resources



Scheduling vs Planning

- Scheduling
  -Decide when and how to perform a given set of actions
    - Time constraints
    - Resource constraints
    - Objective functions
  -Typically **NP-complete**

- Planning
  -Decide what actions to use to achieve some set of objectives
  -Can be much worse than NP-complete



Topics not covered

Many more….