CS 4649/7649  
Robot Intelligence: Planning

Classical Planning Summary

Sungmoon Joo

School of Interactive Computing  
College of Computing  
Georgia Institute of Technology

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 r
    - 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** $\Sigma = (S, A, E, \gamma)$
  - $S = \{\text{abstract states}\}$
    - e.g., states might include a robot's location, but not its position and orientation
  - $A = \{\text{abstract actions}\}$
    - e.g., "move robot from loc2 to loc1" may need complex lower-level implementation
  - $E = \{\text{abstract exogenous events}\}$
    - Not under the agent's control
  - $\gamma = \text{state transition function}$
    - Gives the next state, or possible next states, after an action or event
      - $\gamma: S \times (A \cup E) \rightarrow S$ or $\gamma: S \times (A \cup E) \rightarrow 2^S$
Use ground atoms to represent both rigid and varying properties of a system $\Sigma$

To represent a state $\sigma$ of $\Sigma$
- $s = \{\text{all ground atoms of } L \text{ that are true in } \sigma\}$
- e.g., $s_1 = \{\text{pos}(c_1,\text{loc}_1), \text{loc}(r_1,\text{loc}_1), \text{pos}(c_2,\text{loc}_2), \text{loc}(r_2,\text{loc}_2), \text{adjacent}(\text{loc}_1,\text{loc}_2), \text{adjacent}(\text{loc}_2,\text{loc}_1)\}$

$S = \{\text{all sets of ground atoms of } L\}$
- Call each $s \in S$ a state
- $S$ may contain some "states" that don't actually represent states of $\Sigma$
  - e.g., $\{\text{pos}(c_1,\text{loc}_1), \text{pos}(c_1,\text{loc}_2)\}$
  - Not a big problem if we represent things correctly

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 $p^c$
  - Number of possible states is $2^{pc}$
Classical Planning

Operator vs Action

- Operator
  \( \sigma(x_1, ..., x_k) \)
  - Precond: \( p_1, p_2, ..., p_h \)
  - Effects: \( e_1, e_2, ..., e_h \)

- Action: a ground instance of an operator
  \( \text{take}(r, c) \)
  - Precond: \( \text{loc}(r, c), \text{pos}(c, c), \neg \text{loaded}(r) \)
  - Effects: \( \text{pos}(c, r), \neg \text{pos}(c, c), \text{loaded}(r) \)

Classical Planning

State Transition

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

- Example:
  - \( S = \{ s_0, ..., s_5 \} \)
  - \( A = \{ \text{move1, move2, put, take, load, unload} \} \)
  - \( E = \{ \} \)
  - 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 $\Sigma$
- 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 $\Sigma$
- 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 = $s_5$

Classical Plan: a sequence of actions

\langle \text{take}, \text{move1}, \text{load}, \text{move2} \rangle

Policy: partial function from $S$ into $A$  

\text{i.e. State } \rightarrow \text{Action}

e.g. 
\{(s_0, \text{take}), (s_1, \text{move1}), (s_3, \text{load}), (s_4, \text{move2})\}

- Both, if executed starting at $s_0$, produce $s_3$
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 $P = (Σ, s_0, S_g)$
  - 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
  
  \[ \text{Contains}(Suitcase, Laptop, S0) \]
  
  \[ \text{Working}(Robot, S7) \]
  
- **Actions** are reified functions of constants. (They can be treated as constants themselves)
  
  \[ \text{Put}(Laptop, Suitcase) \quad \text{Open}(Car) \quad \text{Lock}(Car) \]
  
- The do function:

  \[
  do(\alpha, \sigma_0) \rightarrow \sigma_1 \\
  \alpha = \text{action} \\
  \sigma = \text{state}
  \]

*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

\[ \text{In}(Robot, Hall, s) \land \text{Open}(rm, s) = \text{In}(Robot, rm, do(\text{enter}(rm), s)) \]

\[ \text{In}(Robot, Hall, s) \land \text{Open}(rm, s) = \neg \text{In}(Robot, Hall, do(\text{enter}(rm), s)) \]

For each unchanged fluent we add:

\[ \text{Open}(Office, s) \Rightarrow \text{Open}(Office, do(\text{enter}(rm), s)) \]

\[ \neg \text{Open}(Office, s) \Rightarrow \neg \text{Open}(Office, do(\text{enter}(rm), s)) \]

\[ \text{Color}(Sky, Blue, s) \Rightarrow \text{Color}(Sky, Blue, do(\text{enter}(rm), s)) \]

\[ \neg \text{Color}(Sky, Blue, s) \Rightarrow \neg \text{Color}(Sky, Blue, do(\text{enter}(rm), s)) \]

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:

\[ A, B, C, Table \]
\[ IsBlock(A), IsBlock(B) \ldots \]

Ground Literals:

\[ On(B, A), On(C, Table) \]
\[ Clear(B), Clear(Table) \]

Actions:

\[ move(x, y, z) \]

PC:

\[ On(x, y) \]
\[ Clear(x) \]
\[ On(z, z) \]
\[ Clear(z) \]

D:

\[ On(x, y) \]
\[ Clear(x) \]
\[ Clear(y) \]

A:

\[ On(x, z) \]
\[ Clear(y) \]

Domain Axioms:

\[ On(y, x) \land On(z, x) \land (x \neq Table) \implies (y = z) \]

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

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

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 ⊂ PSPACE and NP ⊂ PSPACE,
- But, not whether P ≠ 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
Temporal Planning

If we need an explicit representation of time

- Actions Have Duration
- Timed Conditions and Effects

<table>
<thead>
<tr>
<th>Activity</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Before</td>
</tr>
<tr>
<td>B</td>
<td>Meets</td>
</tr>
<tr>
<td>A</td>
<td>Overlaps</td>
</tr>
<tr>
<td>B</td>
<td>Contains</td>
</tr>
<tr>
<td>A</td>
<td>Starts</td>
</tr>
<tr>
<td>B</td>
<td>Ends</td>
</tr>
<tr>
<td>A</td>
<td>Equals</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>
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....