Artlificial Intelligence

Planning

Christopher Simpkins

Classical Planning

Classical planning is the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment. Two previous approaches:

- Graph search, e.g., A^*
- Hybrid propositional logical agent

Two limitations:

- Require ad-hoc heuristics
- Require explicit representation of exponentially large state space.

Planning Domain Definition Language solves these problems using a factored representation based on first-order logic.

- A **state** is a conjunction of ground atomic fluents – single predicates containing no variables.
  - $At(Truck_1, Melbourne)$ is a ground atomic fluent, $At(t_1, from)$ is not.
- PDDL uses **database semantics**, or the **closed-world assumption**: any fluents not mentioned are false, and unique names represent distinct objects.

KENNESAW STATE UNIVERSITY

Planning Domain Definition Language (PDDL)

Action schema is a family of ground actions.

- Action name and list of variables
- Precondition: conjunction of literals
  - Action a is **applicable** in state s if $s \models a.precondition$
- Effect: conjunction of literals
  - **Result** of executing action a in state s is s' is applying delete list and add list to s:
    - $DEL(a)$, delete list: remove negative literals in action's effects.
    - $ADD(a)$, add list: add positive literals in action's effects.

Action schema:

$$\begin{aligned}
&Action(Fly(p, from, to),\\
&\quad PRECOND: At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)\\
&\quad EFFECT: \neg At(p, from) \wedge At(p, to))
\end{aligned}$$

Ground (variable-free) action:

$$\begin{aligned}
&Action(Fly(P_1, SFO, JFK),\\
&\quad PRECOND: At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)\\
&\quad EFFECT: \neg At(P_1, SFO) \wedge At(P_1, JFK))
\end{aligned}$$

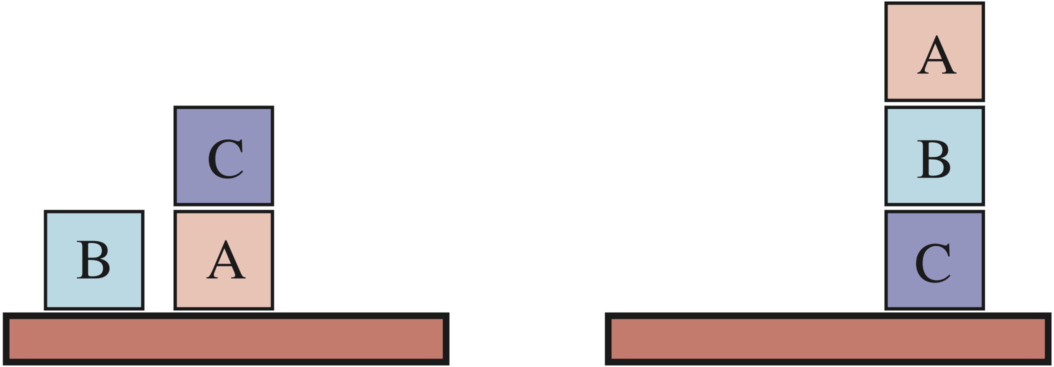
KENNESAW STATE UNIVERSITY

Air Cargo Transport

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\qquad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\qquad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

KENNESAW STATE UNIVERSITY

Blocks World



Start State Goal State

Blocks World PDDL

$$
\begin{aligned}
&Init(On(A, Table) \wedge On(B, Table) \wedge On(C, A) \\
&\quad \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C) \wedge Clear(Table)) \\
&Goal(On(A, B) \wedge On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \textsc{Precond}: On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge \\
&\qquad\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y), \\
&\quad \textsc{Effect}: On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \textsc{Precond}: On(b, x) \wedge Clear(b) \wedge Block(b) \wedge Block(x), \\
&\quad \textsc{Effect}: On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))
\end{aligned}
$$

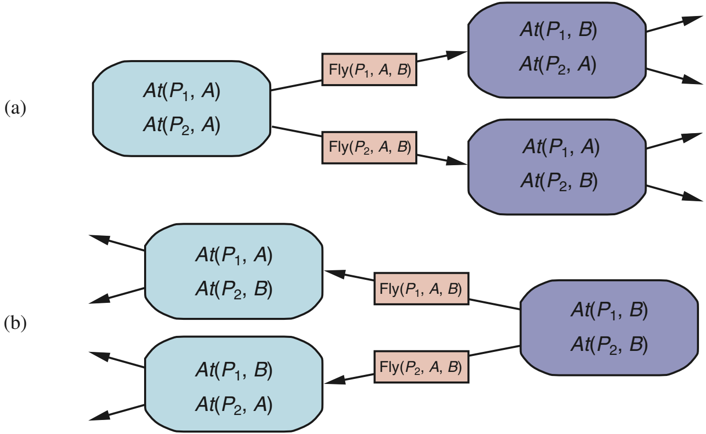
KENNESAW STATE UNIVERSITY

Classical Planning Algorithms

- Forward state space search
- Backward state space search
- SATPlan – boolean satisfiability planning
  - Translate PDDL into propositional form, use a SAT solver
- Graphplan
  - Encode constraints related to preconditions and effects in a **planning graph**.
- Situation calculus
- Constraint satisfaction
- Partial-order planning
  - $Remove(Spare, Trunk)$ and $Remove(Flat, Axle)$ must come before $PutOn(Spare, Axle)$, but removals can happen in any order.

Forward and Backward State Space Planning

- Forward search: unify current state against preconditions of each action schema – **applicable** actions.
- Backward search: unify goal states against effects of action schemas – **relevant** actions.



Hierarchical Planning

Hierarchical task network plans are built from:

- primitive actions, and
- high-level actions (HLA).

HLAs have one or more **refinements**.

- Refinements may contain other HLAs.
- A refinement with only primitive actions is an **implementation**.
- An HLA achieves a goal if at least one of its implementations achieves the goal.

Here are two goal-achieving implementations for the $Go(Home, SFO)$ HLA:

$$
\begin{array}{l}
\mathit{Refinement}(\mathit{Go}(\mathit{Home}, \mathit{SFO}), \\
\quad \text{STEPS: } [\mathit{Drive}(\mathit{Home}, \mathit{SFOLongTermParking}), \\
\qquad\qquad \mathit{Shuttle}(\mathit{SFOLongTermParking}, \mathit{SFO})]\ ) \\
\mathit{Refinement}(\mathit{Go}(\mathit{Home}, \mathit{SFO}), \\
\quad \text{STEPS: } [\mathit{Taxi}(\mathit{Home}, \mathit{SFO})]\ )
\end{array}
$$

Refinements can be produced recursivley, as shown in this vacuum world navigation example:

$$
\begin{array}{l}
\mathit{Refinement}(\mathit{Navigate}([a,b],[x,y]), \\
\quad \text{PRECOND: } a = x \ \wedge \ b = y \\
\quad \text{STEPS: } [\,]\ ) \\
\mathit{Refinement}(\mathit{Navigate}([a,b],[x,y]), \\
\quad \text{PRECOND:} \mathit{Connected}([a,b],[a-1,b]) \\
\quad \text{STEPS: } [\mathit{Left}, \mathit{Navigate}([a-1,b],[x,y])]\ ) \\
\mathit{Refinement}(\mathit{Navigate}([a,b],[x,y]), \\
\quad \text{PRECOND:} \mathit{Connected}([a,b],[a+1,b]) \\
\quad \text{STEPS: } [\mathit{Right}, \mathit{Navigate}([a+1,b],[x,y])]\ ) \\
\ldots
\end{array}
$$

KENNESAW STATE UNIVERSITY

Closing Thoughts

- Fun to create toy worlds and solve them.
  - Look up "Monkey and bananas" problem.
- Still have knowledge-acquisition bottleneck.
- Still have problem of specifying large number of rules and facts for non-trivial problems.
- Still have problem of uncertainty – nondeterministic actions and partial observability.

In rest of course, we address these issues with uncertain reasoning and machine learning.

KENNESAW STATE UNIVERSITY