Lecture 20 of 42

Introduction to Classical Planning: STRIPS & Partial-Order Planning (POP)

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course page: http://snipurl.com/v9v3
Course web site: http://www.kddresearch.org/Courses/CIS730
Instructor home page: http://www.cis.ksu.edu/~bhsu

Reading for Next Class:
Partial plan: http://en.wikipedia.org/wiki/Partial_plan

Lecture Outline
- Reading for Next Class: Section 11.3 (p. 387 – 394), R&N 2nd edition
- Last Class: Knowledge Representation Concluded; Midterm Review
  - Inheritance semantics
  - Midterm exam emphasis
    - Rational intelligent agents: reflex, reflex/state, goals, preferences
    - Search: heuristic, constraint, game tree
    - Knowledge representation and inference: logic, resolution; FC/BC, $L_{SAT}$
  - Planning problem defined
    - Initial conditions
    - Actions: preconditions, postconditions
    - Goal conditions / goal test
  - Limitations of situation calculus and FOL
  - STRIPS operators: represent actions with preconditions, ADD/DELETE lists
- Coming Week: Midterm; More Classical and Robust Planning
PLANNING IN SITUATION CALCULUS

PlanResult(p, s) is the situation resulting from executing p in s
PlanResult([], s) = s
PlanResult([a], s) = PlanResult(p, Result(a, s))

Initial state: At(Home, S0) ∧ ¬Have(Milk, S0) ∧ ...

Actions as Successor State axioms
Have(Milk, Result(a, s)) ⇔
[(a = Buy(Milk) ∧ At(Supermarket, s)) ∨ (Have(Milk, s) ∧ a ≠ ...)]

Query
s = PlanResult(p, S0) ∧ At(Home, s) ∧ Have(Milk, s) ∧ ...

Solution
p = [Go(Supermarket), Buy(Milk), Buy(Bananas), Go(HWS),...]

Principal difficulty: unconstrained branching, hard to apply heuristics

MAKING PLANS USING FOL:
Review

Initial condition in KB:
At(Agent, [1,1], S0)
At(Gold, [1,2], S0)

Query: Ask(KB, ∃s Holding(Gold, s))
i.e., in what situation will I be holding the gold?

Answer: {s/Result(Grab, Result(Forward, S0))}
i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S0 and that S0 is the only situation described in the KB

Making Plans — Better Way: Review

Represent plans as action sequences \([a_1, a_2, \ldots, a_n]\)

PlanResult\((p, s)\) is the result of executing \(p\) in \(s\)

Then the query \(\text{ASK}(KB, \exists p \textit{ Holding}(Gold, \text{PlanResult}(p, S_0)))\)
has the solution \([p/\text{[Forward, Grab]}]\)

Definition of PlanResult in terms of Result:
\[
\forall s \textit{ PlanResult}([], s) = s \\
\forall a, p, s \textit{ PlanResult}([a|p], s) = \text{PlanResult}(p, \text{Result}(a, s))
\]

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner.

---

STRIPS Operators

Tidily arranged actions descriptions, restricted language

**ACTION:** Buy\((x)\)

**PRECONDITION:** At\((p)\), In\((p, x)\)

**EFFECT:** Have\((x)\)

[Note: this abstracts away many important details!]

Restricted language \(\Rightarrow\) efficient algorithm

- Precondition: conjunction of positive literals
- Effect: conjunction of literals

\[\text{At}(p) \land \text{In}(p, x)\]

\[
\begin{align*}
\text{Buy}(x) \\
\text{Have}(x)
\end{align*}
\]

\[\text{Action}(\text{Fly}(p, \text{from}, \text{to}), \text{PRECOND: At}(p, \text{from}) \land \text{Plane}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to}) \land \text{EFFECT:} -\text{At}(p, \text{from}) \land \text{At}(p, \text{to}))\]

**State Space versus Plan Space**

Standard search: node = concrete world state  
Planning search: node = partial plan  

Defn: open condition is a precondition of a step not yet fulfilled  

Operators on partial plans:  
- add a link from an existing action to an open condition  
- add a step to fulfill an open condition  
- order one step wrt another  

Gradually move from incomplete/vague plans to complete, correct plans

---

**Air Cargo Transport Problem**

**STRIPS Specification**

```
Init(At(C1, SFO) \ At(C2, JFK) \ At(P1, SFO) \ At(P2, JFK) 
  \ Cargo(C1) \ Cargo(C2) \ Plane(P1) \ Plane(P2) 
  \ Airport(JFK) \ Airport(SFO))
Goal(At(C1, JFK) \ At(C2, SFO))
Action(Load(c, p, a),  
  Precond: At(c, a) \ At(p, a) \ Cargo(c) \ Plane(p) \ Airport(a)  
  Effect: ~At(c, a) \ ~In(c, p))
Action(Unload(c, p, a),  
  Precond: In(c, p) \ At(p, a) \ Cargo(c) \ Plane(p) \ Airport(a)  
  Effect: At(c, a) \ ~In(c, p))
Action(Fly(p, from, to),  
  Precond: At(p, from) \ Plane(p) \ Airport(from) \ Airport(to)  
  Effect: ~At(p, from) \ At(p, to))
```

Figure 11.2  
p. 380 R&N 2°
STRIPS and Its Limitations:
Need for Richer Planning Language

- **What STRIPS Can Represent**
  - **States**
  - **Goals**
  - **Actions (using action schema)**
    - **Preconditions**: must be true before action can be applied
    - **Effects**: asserted afterwards
  - **Real STRIPS**: ADD, DELETE Lists for Operators
  - **STRIPS Assumption**
    - Representational frame problem solution
    - Default is that conditions remain unchanged unless mentioned in effect
  - **What STRIPS Cannot Represent**
    - Negated preconditions
    - Inequality constraints

- **Richer Planning Language: Action Description Language (ADL)**

---

**Figure 11.1**

<table>
<thead>
<tr>
<th>STRIPS Language</th>
<th>ADL Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only positive literals in states:</td>
<td>Positive and negative literals in states:</td>
</tr>
<tr>
<td>Poor &amp; Unknown</td>
<td>~Rich &amp; ~Famous</td>
</tr>
<tr>
<td>Closed World Assumption:</td>
<td>Open World Assumption:</td>
</tr>
<tr>
<td>Unmentioned literals are false.</td>
<td>Unmentioned literals are unknown.</td>
</tr>
<tr>
<td>Effect ( P \land \neg Q ) means add ( P ) and delete ( Q ).</td>
<td>Effect ( P \land \neg Q ) means add ( P ) and delete ( \neg Q ).</td>
</tr>
<tr>
<td>Only ground literals in goals:</td>
<td>Quantified variables in goals:</td>
</tr>
<tr>
<td>Rich &amp; Famous</td>
<td>( 3x (P(x)) \land (P(y)) ) is the goal of having ( P_1 ) and ( P_2 ) in the same place.</td>
</tr>
<tr>
<td>Goals are conjunctions.</td>
<td>Goals allow conjunction and disjunction:</td>
</tr>
<tr>
<td>Rich &amp; Famous</td>
<td>~Poor &amp; (Famous &amp; Smart)</td>
</tr>
<tr>
<td>Effects are conjunctions.</td>
<td>Conditional effects allowed:</td>
</tr>
<tr>
<td></td>
<td>when ( P \cdot E ) means ( E ) is an effect</td>
</tr>
<tr>
<td></td>
<td>only if ( P ) is satisfied.</td>
</tr>
<tr>
<td>No support for equality.</td>
<td>Equality predicate ( (x = y) ) is built in.</td>
</tr>
<tr>
<td>No support for types.</td>
<td>Variables can have types, as in ( y : Plane ).</td>
</tr>
</tbody>
</table>

Simple Spare Tire Problem [1]: Illustrated Example

START
~At(Spare) Intact(Spare) On(Spare)
On(Flat) Flat(Flat)

Finish


Simple Spare Tire Problem [2]: ADL Specification

Init(At(Flat, Axle) ∧ At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(At(Flat, Axle), Remove(Spare, Trunk),
PRECOND: At(Spare, Trunk)
EFFECT: ~At(Spare, Trunk) ∧ At(Flat, Axle))
Action(At(Flat, Axle), PutOn(Spare, Axle),
PRECOND: At(Spare, Ground) ∧ ~At(Flat, Axle)
EFFECT: ~At(Spare, Ground) ∧ At(Spare, Axle))
Action(LeaveOvernight,
PRECOND: ~At(Spare, Ground) ∧ ~At(Spare, Axle)
∧ ~At(Flat, Ground) ∧ ~At(Flat, Axle))

Figure 11.3
p. 381 R&N 2°

**Blocks World: Three-Block Tower Problem**

```prolog
Init(On(A, Table) \(\land\) On(B, Table) \(\land\) On(C, Table) \\
\(\land\) Block(A) \(\land\) Block(B) \(\land\) Block(C) \\
\(\land\) Clear(A) \(\land\) Clear(B) \(\land\) Clear(C))
Goal(On(A, B) \(\land\) On(B, C))
Action(Move(b, x, y), \\
  PRECOND: On(b, x) \(\land\) Clear(b) \(\land\) Clear(y) \(\land\) Block(b) \(\land\) \\
  \(b \neq x\) \(\land\) \(b \neq y\) \(\land\) \(x \neq y\),
  EFFECT: On(b, y) \(\land\) Clear(x) \(\land\) \(\neg\) On(b, x) \(\land\) \(\neg\) Clear(y))
Action(MoveToTable(b, x), \\
  PRECOND: On(b, x) \(\land\) Clear(b) \(\land\) Block(b) \(\land\) \(b \neq x\),
  EFFECT: On(b, Table) \(\land\) Clear(x) \(\land\) \(\neg\) On(b, x))
```

---

**Forward (Progression) vs. Backward (Regression) State Space Search**

(a) ![Diagram](image)

(b) ![Diagram](image)

---

FAILURE OF NON-INTERLEAVED PLANNING:
SUSSMAN ANOMALY IN BLOCKS WORLD

Search versus Planning: State Space Search

Consider the task get milk, bananas, and a cordless drill
Standard search algorithms seem to fail miserably:

After-the-fact heuristic/goal test inadequate
**Partial Order Planning (POP) [1]:**

**Total Order Plans & Interleavings**

![Diagram of Partial Order Plans vs. Total Order Plans](image)

*Figure 11.6 p. 389 R&N 2*

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.

---

**Partial Order Planning (POP) [2]:**

**Definition — Complete Plans**

![Diagram of Complete Plans](image)

A plan is complete iff every precondition is achieved.

A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it.

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.
**POP Algorithm [1]:
Top-Level Functions**

```plaintext
function POP(initial, goal, operators) returns plan
    plan ← MAKE-MINIMAL-PLAN(initial, goal)
    loop do
        if SOLUTION?(plan) then return plan
        S_next ← SELECT-NEXT(plan)
        Choose-Operator(plan, operators, S_next, c)
        REMOVE-THREATS(plan)
    end

function SELECT-NEXT(plan) returns S_next:
    pick a plan step S_next from STEP(plan)
    with a precondition c that has not been achieved
    return S_next, c
```

**POP Algorithm [2]:
Lower-Level Functions & Properties**

```plaintext
procedure Choose-Operator(plan, operators, S_next, c)
    choose a step S_next from operators or STEP(plan) that has c as an effect
    if there is no such step then fail
    add the causal link S_next → c to LEVEL(plan)
    add the ordering constraint S_next < S_next to ORDERING(plan)
    if S_next is a newly added step from operators then
        add S_next to STEP(plan)
        add S_next < S_next to ORDERING(plan)

procedure REMOVE-THREATS(plan)
    for each S_next that threatens a link S_i → S_j in LEVEL(plan) do
        choose either
        Detection: Add S_next < S_i to ORDERING(plan)
        Prevention: Add S_j < S_next to ORDERING(plan)
        if not CONFLICTS(plan) then fail
    end
```

POP is sound, complete, and systematic (no repetition)

Extensions for disjunction, universals, negation, conditionals
Clobbering and Promotion / Demotion

A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., Go(Home) clobbers At(HWS):

- Demotion: put before Go(HWS)
- Promotion: put after Buy(Drill)

Preview: How Things Go Wrong in Planning

Incomplete information
- Unknown preconditions, e.g., Intact(Spare)?
- Disjunctive effects, e.g., Inflated(x) causes Inflated(x) \lor SlowHiss(x) \lor Burst(x) \lor BrokenPump \lor \ldots

Incorrect information
- Current state incorrect, e.g., spare NOT intact
- Missing/incorrect postconditions in operators

Qualification problem:
- can never finish listing all the required preconditions and possible conditional outcomes of actions

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.
Terminology

- Classical Planning – STRIPS and ADL
  - Planning problem defined
    - Initial conditions
    - Actions: preconditions, effects (postconditions)
    - Goal conditions / goal test
  - STRIPS operators: action specifications
  - ADL operators: allow negated preconditions, inequality

- Partial-Order Planning
  - Represent multiple possible interleavings
  - Keep track of which ones are achievable
  - Complete plans
    - Every precondition achieved,
    - No clobberings by possibly intervening steps

- Sussman Anomaly
  - Contains threat that needs to be resolved to get to goal
  - Illustrates need for partial-order planning, promotion / demotion

Summary Points

- Last Class: Knowledge Representation Concluded; Midterm Review
  - Inheritance semantics
  - Midterm emphasis: intelligent agents, search, KR, resolution/unification

- Today: Classical Planning – STRIPS and ADL
  - Planning problem defined
    - Initial conditions
    - Actions: preconditions, postconditions
    - Goal conditions / goal test
  - Limitations of situation calculus and FOL
  - STRIPS operators
  - ADL operators: allow negated preconditions, inequality

- Next Time (After Exam): More Classical and Robust Planning
  - Hierarchical abstraction planning (ABSTRIPS)
  - Robust planning: sensorless, conditional, monitoring/replanning, continual

- Coming Week: Midterm; Planning Continued