Instructor: Sofus A. Macskassy, macskass@usc.edu
TAs: Nadeesha Ranashinghe (nadeeshr@usc.edu)
     William Yeoh (wyeoh@usc.edu)
     Harris Chiu (chiciu@usc.edu)

Lectures: MW 5:00-6:20pm, OHE 122 / DEN
Office hours: By appointment
Class page: http://www-rcf.usc.edu/~macskass/CS561-Spring2010/

This class will use http://www.uscden.net/ and class webpage
- Up to date information
- Lecture notes
- Relevant dates, links, etc.

Course material:
Planning (AIMA Ch. 11)

- Search vs. planning
- STRIPS operators
- Partial-order planning
What we have so far

- Can TELL KB about new percepts about the world
- KB maintains model of the current world state
- Can ASK KB about any fact that can be inferred from KB

How can we use these components to build a planning agent?

i.e., an agent that constructs plans that can achieve its goals, and that then executes these plans?
Remember: Problem-Solving Agent

function **SIMPLE-PROBLEM-SOLVING-AGENT**( percept) returns an action
static:

seq, an action sequence, initially empty
state, some description of the current world state
goal, a goal, initially null
problem, a problem formulation

state $\leftarrow$ UPDATE-STATE(state, percept)  // What is the current state?
if seq is empty then

  goal $\leftarrow$ FORMULATE-GOAL(state)  // From LA to San Diego (given curr. state)
  problem $\leftarrow$ FORMULATE-PROBLEM(state, goal)  // e.g., Gas usage
  seq $\leftarrow$ SEARCH(problem)
  action $\leftarrow$ RECOMMENDATION(seq, state)
  seq $\leftarrow$ REMAINDER(seq, state)  // If fails to reach goal, update

return action

Note: This is *offline* problem-solving. *Online* problem-solving involves acting w/o complete knowledge of the problem and environment.
Simple planning agent

- Use percepts to build model of current world state
- IDEAL-PLANNER: Given a goal, algorithm generates plan of action
- STATE-DESCRIPTION: given percept, return initial state description in format required by planner
- MAKE-GOAL-QUERY: used to ask KB what next goal should be
A Simple Planning Agent

function SIMPLE-PLANNING-AGENT(percept) returns an action

static: KB, a knowledge base (includes action descriptions)
p, a plan (initially, NoPlan)
t, a time counter (initially 0)

local variables: G, a goal
                current, a current state description

TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
current ← STATE-DESCRIPTION(KB, t)

if p = NoPlan then
    G ← ASK(KB, MAKE-GOAL-QUERY(t))
p ← IDEAL-PLANNER(current, G, KB)

if p = NoPlan or p is empty then
    action ← NoOp
else
    action ← FIRST(p)
    p ← REST(p)
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t+1

return action

Like popping from a stack
Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*

Standard search algorithms seem to fail miserably:

After-the-fact heuristic/goal test inadequate
Search vs. planning

Planning systems do the following:
1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>Lisp data structures</td>
<td>Logical sentences</td>
</tr>
<tr>
<td>Actions</td>
<td>Lisp code</td>
<td>Preconditions/outcomes</td>
</tr>
<tr>
<td>Goal</td>
<td>Lisp code</td>
<td>Logical sentence (conjunction)</td>
</tr>
<tr>
<td>Plan</td>
<td>Sequence from $S_0$</td>
<td>Constraints on actions</td>
</tr>
</tbody>
</table>
Planning in situation calculus

\( \text{PlanResult}(p, s) \) is the situation resulting from executing \( p \) in \( s \)

\[
\text{PlanResult}([], s) = s \\
\text{PlanResult}([a | p], s) = \text{PlanResult}(p, \text{Result}(a, s))
\]

Initial state \( \text{At(Home, } S_0) \wedge \neg \text{Have(Milk, } S_0) \wedge \ldots \)

Actions as Successor State axioms
\( \text{Have(Milk, Result}(a, s)) \iff \]
\[
[(a = \text{Buy(Milk)} \wedge \text{At(Supermarket, } s)) \vee (\text{Have(Milk, } s) \wedge a \neq \ldots\text{})]
\]

Query
\[
s = \text{PlanResult}(p, S_0) \wedge \text{At(Home, } s) \wedge \text{Have(Milk, } s) \wedge \ldots
\]

Solution
\[
p = [\text{Go(Supermarket)}, \text{Buy(Milk)}, \text{Buy(Bananas)}, \text{Go(HWS)}, \ldots]
\]

Principal difficulty: unconstrained branching, hard to apply heuristics
Basic representation for planning

- Most widely used approach: uses STRIPS language

- **States**: conjunctions of function-free ground literals (I.e., predicates applied to constant symbols, possibly negated); e.g.,
  
  \[ \text{At(Home)} \land \neg \text{Have(Milk)} \land \neg \text{Have(Bananas)} \land \neg \text{Have(Drill)} \ldots \]

- **Goals**: also conjunctions of literals; e.g.,
  
  \[ \text{At(Home)} \land \text{Have(Milk)} \land \text{Have(Bananas)} \land \text{Have(Drill)} \]

  but can also contain variables (implicitly universally quant.); e.g.,

  \[ \text{At}(x) \land \text{Sells}(x, \text{Milk}) \]
Planner vs. theorem prover

- **Planner**: ask for sequence of actions that makes goal true if executed

- **Theorem prover**: ask whether query sentence is true given KB
STRIPS operators

Tidily arranged actions descriptions, restricted language

**ACTION:** $Buy(x)$

**PRECONDITION:** $At(p), Sells(p, x)$

**EFFECT:** $Have(x)$

[Note: this abstracts away many important details!]

Restricted language $\Rightarrow$ efficient algorithm

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

Graphical notation:

```
At(p) Sells(p, x)

Buy(x)

Have(x)
```
STRIPS cont’d

- Every literal not mentioned in effect remains unchanged
  → Circumvents the frame problem mentioned earlier
STRIPS vs. Action Desc. Lang (ADL)

STRIPS is not always adequate for all real-world domains and other planning languages have been developed to address some deficiencies.

<table>
<thead>
<tr>
<th>STRIPS Language</th>
<th>ADL Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only positive literals in state: Poor ∧ Unknown</td>
<td>Positive and negative literals ¬Rich ∧ ¬Famous</td>
</tr>
<tr>
<td>Closed World Assumption: Unmentioned literals are false</td>
<td>Open World Assumption: Unmentioned literals are unknown</td>
</tr>
<tr>
<td>Effect P ∧ ¬Q means add P and delete Q</td>
<td>Effect P ∧ ¬Q means add P and ¬Q and delete ¬P and Q</td>
</tr>
<tr>
<td>Only ground literals in goals</td>
<td>Quantified variables in goals</td>
</tr>
<tr>
<td>Goals are conjunctions</td>
<td>Goals allow disjuncts + conjuncts</td>
</tr>
<tr>
<td>Effects are conjunctions</td>
<td>Conditional effects allowed</td>
</tr>
<tr>
<td>No support for equality</td>
<td>Equality built in</td>
</tr>
<tr>
<td>No support for types</td>
<td>Variables can have types</td>
</tr>
</tbody>
</table>
Comparison of ‘fly’ action definitions

**STRIPS:**
- Action(Fly(p,from,to),
  Precond:   At(p,from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
  Effect:    ~At(p,from) ∧ At(p,to))

**ADL:**
- Action(Fly(p: Plane, from: Airport, to: Airport),
  Precond:   At(p,from) ∧ (from ≠ to)
  Effect:    ~At(p,from) ∧ At(p,to))
Example STRIPS problem

- 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)
**Example STRIPS problem**

- **Init**:
  \[\text{At}(C1, SFO) \land \text{At}(C2, JFK) \land \text{At}(P1, SFO) \land \text{At}(P2, JFK) \land \text{Cargo}(C1) \land \text{Cargo}(C2) \land \text{Plane}(P1) \land \text{Plane}(P2) \land \text{Airport}(JFK) \land \text{Airport}(SFO)\]

- **Goal**:
  \[\text{At}(C1, JFK) \land \text{At}(C2, SFO)\]

- **Action** \(\text{Load}(c,p,a)\):
  - **Precond**: \(\text{At}(c,a) \land \text{At}(p,a) \land \text{Cargo}(c) \land \text{Plane}(p) \land \text{Airport}(a)\)
  - **Effect**: \(!\text{At}(c,a) \land \text{In}(c,p)\)

- **Action** \(\text{Unload}(c,p,a)\):
  - **Precond**: \(\text{In}(c,p) \land \text{At}(p,a) \land \text{Cargo}(c) \land \text{Plane}(p) \land \text{Airport}(a)\)
  - **Effect**: \(\text{At}(c,a) \land !\text{In}(c,p)\)

- **Action** \(\text{Fly}(p,\text{from},\text{to})\):
  - **Precond**: \(\text{At}(p,\text{from}) \land \text{Plane}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to})\)
  - **Effect**: \(!\text{At}(p,\text{from}) \land \text{At}(p,\text{to})\)

**One possible solution:**

\[
\begin{array}{c}
\text{Load}(C1,P1,SFO) \\
\text{Fly}(P1,SFO,JFK) \\
\text{Unload}(C1,P1,JFK) \\
\text{Load}(C2,P2,JFK) \\
\text{Fly}(P2,JFK,SFO) \\
\text{Unload}(C2,P2,SFO)
\end{array}
\]
Types of planners

- Situation space planner: search through possible situations

- Progression planner: start with initial state, apply operators until goal is reached
  
  Problem: high branching factor!

- Regression planner: start from goal state and apply operators until start state reached
  
  Why desirable? usually many more operators are applicable to
  initial state than to goal state.
  Difficulty: when want to achieve a conjunction of goals

Initial STRIPS algorithm: situation-space regression planner
State space vs. 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
Operations on plans

- Refinement operators: add constraints to partial plan

- Modification operator: every other operators
Types of planners

- **Partial order planner**: some steps are ordered, some are not

- **Total order planner**: all steps ordered (thus, plan is a simple list of steps)

- **Linearization**: process of deriving a totally ordered plan from a partially ordered plan.
Partially ordered plans

How many total-order plans are there to get both shoes on?

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.
Example STRIPS problem

- 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))

One possible solution:

[Load(C1,P1,SFO) ^ In(C1,P1) ^ Cargo(C1) ^ Plane(P1) ^ Airport(SFO) ^ At(C1, SFO) ^ At(P1, SFO) ^ At(C2, JFK) ^ At(P2, JFK) ^ Cargo(C2) ^ Plane(P2) ^ Airport(JFK) ^ Airport(SFO)]

Fly(P1,SFO,JFK)
Unload(C1,P1,JFK)
Fly(P2,JFK,SFO)
Unload(C2,P2,JFK)
Example STRIPS problem

- 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))

One possible solution:

```
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, JFK)
```

One possible solution:

```
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, JFK)
```
Plan

We formally define a plan as a data structure consisting of:

- Set of plan steps (each is an operator for the problem)
- Set of step ordering constraints
  
  e.g., $A \parallel B$ means “A before B”

- Set of variable binding constraints
  
  e.g., $v = x$ where $v$ variable and $x$ constant or other variable

- Set of causal links
  
  e.g., $A \xrightarrow{c} B$ means “A achieves $c$ for B”
function POP(initial, goal, operators) returns plan

    plan ← Make-Minimal-Plan(initial, goal)
    loop do
        if Solution?(plan) then return plan
        S_{need}, c ← Select-Subgoal(plan)
        Choose-Operator(plan, operators, S_{need}, c)
        Resolve-Threats(plan)
    end

function Select-Subgoal(plan) returns S_{need}, c

    pick a plan step S_{need} from STEPS(plan)
    with a precondition c that has not been achieved

    return S_{need}, c
**POP algorithm (cont.)**

```
procedure CHOOSE-OPERATOR(plan, operators, S_{need}, c)
    choose a step S_{add} from operators or STEPS(plan) that has c as an effect
    if there is no such step then fail
    add the causal link S_{add} \rightarrow c \rightarrow S_{need} to LINKS(plan)
    add the ordering constraint S_{add} < S_{need} to ORDERINGS(plan)
    if S_{add} is a newly added step from operators then
        add S_{add} to STEPS(plan)
        add Start < S_{add} < Finish to ORDERINGS(plan)
```

```
procedure RESOLVE-THREATS(plan)
    for each S_{threat} that threatens a link S_i \rightarrow c \rightarrow S_j in LINKS(plan) do
        choose either
        Demotion: Add S_{threat} < S_i to ORDERINGS(plan)
        Promotion: Add S_j < S_{threat} to ORDERINGS(plan)
        if not CONSISTENT(plan) then fail
    end
```

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

Extensions for disjunction, universals, negation, conditionals
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 \( Buy(Drill) \)
STRIPS: block world toy problem

- Init(On(A,Table) ^ On(B,Table) ^ On(C, Table)
  ^ Block(A) ^ Block(B) ^ Block(C)
  ^ Clear(A) ^ Clear(B) ^ Clear(C))

- Goal(On(A,B) ^ On(B,C))

- Action(PutOn(x, y),
  Precond:  On(x,z) ^ Clear(x) ^ Clear(y) ^ Block(x) ^ Block(y) ^
  (x≠z) ^ (x≠y) ^ (y≠z)
  Effect:   On(x,y) ^ Clear(z) ^ ~On(x,z) ^ ~Clear(y))

- Action(PutOnTable(x),
  Precond:  On(x,z) ^ Clear(x) ^ Block(x) ^ (x≠z)
  Effect:   On(x,Table) ^ Clear(z) ^ ~On(x,z))
STRIPS: block world toy problem

- **Init**:\( \text{On(A,Table)} \land \text{On(B,Table)} \land \text{On(C, Table)} \land \text{Block(A)} \land \text{Block(B)} \land \text{Block(C)} \land \text{Clear(A)} \land \text{Clear(B)} \land \text{Clear(C)} \)

- **Goal**: \( \text{On(A,B)} \land \text{On(B,C)} \)

- **Action** (\text{PutOn}(x, y))
  - **Precond**: \( \text{On(x,z)} \land \text{Clear(x)} \land \text{Block(x)} \land (x\neq z) \land (x\neq y) \land (y\neq z) \)
  - **Effect**: \( \text{On(x,y)} \land \text{Clear(z)} \land \neg \text{On(x,z)} \land \neg \text{Clear(y)} \)

- **Action** (\text{PutOnTable}(x))
  - **Precond**: \( \text{On(x,z)} \land \text{Clear(x)} \land \text{Block(x)} \land (x\neq z) \)
  - **Effect**: \( \text{On(x,Table)} \land \text{Clear(z)} \land \neg \text{On(x,z)} \)

**One possible solution:**

\[
\begin{align*}
\text{PutOn}(B,C) \\
\text{PutOn}(A,B)
\end{align*}
\]

Order is very important! Cannot do “PutOn(A,B)” first...!
“Sussman anomaly” problem

Start State

Goal State

Goal Conditions:
On(A,B)
On(B,C)
"Sussman anomaly" problem (cont.)

Start State

Goal State

Goal Conditions:
On(A,B)
On(B,C)

Pursue: On(A,B)
“Sussman anomaly” problem (cont.)

Start State

Goal State

Goal Conditions:
On(A,B)
On(B,C)

Pursue: On(B,C)
POP Solution... (cont.)

START
On(C,A) On(A,Table) Cl(B) Cl(C) On(B,Table)

FINISH
POP Solution... (cont.)

START

On(C,A) On(A,Table) Cl(B) Cl(C) On(B,Table)

PutOn(B,C)

~Cl(C) On(B,C) ~On(B,z)

On(A,B) On(B,C)

FINISH
POP Solution... (cont.)

START
On(C, A) On(A, Table) Cl(B) Cl(C) On(B, Table)

PutOn(A, B)
Clobbers Cl(B)
⇒ Order after PutOn(B, C)

FINISH
START

On(C,A) On(A,Table) Cl(B) Cl(C) On(B,Table)

PutOnTable(C)

On(C,Table) Cl(z) ~On(C,z)

PutOn(A,B)

~On(A,z) On(A,B) ~Cl(B)

On(A,B) On(B,C)

FINISH

PutOn(A,B)
Clobbers Cl(B)
⇒ Order after PutOn(B,C)

PutOn(B,C)
Clobbers Cl(C)
⇒ Order after PutOnTable(C)

On(C,Table) Cl(z) ~On(C,z)

Cl(A) On(A,z) Cl(B)

Cl(B) Cl(C) On(B,z)

PutOn(B,C)

~Cl(C) On(B,C) ~On(B,z)