Planning

A planning agent will construct plans to achieve its goals, and then execute them.

Analyze a situation in which it finds itself and develop a strategy for achieving the agent’s goal.

Achieving a goal requires finding a sequence of actions that can be expected to have the desired outcome.

Problem Solving

Representation of actions – actions generate successor states

Representation of states – all state representations are complete

Representation of goals – contained in goal test and heuristic function

Representation of plans – unbroken sequence of actions leading from initial to goal state.

Planning Example

GOAL: Get a quart of milk and a bunch of bananas and a variable-speed cord-less drill.

Planning Versus Problem Solving

(1) Open up the representation of states, goals and actions.

- States and goals represented by sets of sentences – \textit{Have(Milk)}
- Actions represented by rules that represent their preconditions and effects: \textit{Buy(x)} achieves \textit{Have(x)}

This allows the planner to make direct connections between states and actions.
Planning Versus Problem Solving

(2) Planner is free to add actions to the plan wherever they are needed, rather than in an incremental sequence starting at the initial state.

- No connection between the order of planning and the order of execution.
- Representation of states as sets of logical sentences makes this freedom possible.

Planning Versus Problem Solving

(3) Most parts of the world are independent of most other parts.

- Can solve $Have(Milk) \land Have(Bananas) \land Have(Drill)$ using divide-and-conquer strategy.
- Can re-use subplans (go to supermarket)

Planning as a Logical Inference Problem

Axioms:
On(A,C), On(C,Table), On(D,B), On(B,Table), Clear(A), Clear(D)
plus rules for moving things around...
Prove: On(A,B) \land On(B,C)

Planning as Deduction: Situation Calculus

In first-order logic, once a statement is shown to be true, it remains true forever.

**Situation calculus:** way to describe change in first-order logic.
Situation Calculus

**fluenets:** functions and predicates that vary from one situation to the next

\[
\begin{align*}
on(A, C) & \quad on(A, C, S_0) \\
at(agent, [1, 1]) & \quad at(agent, [1, 1], S_0)
\end{align*}
\]

**atemporal** functions and predicates are also allowed

\[
\begin{align*}
block(A) \\
gold(G_1)
\end{align*}
\]

Situation Calculus: Actions

Actions are described by stating their effects.

**possibility axiom:** preconditions \( \Rightarrow \) Poss\((a, s)\).

\[
\forall s \forall x \neg On(x, Table, s) \land Clear(x, s) \Rightarrow Poss(PlaceOnTable(x, s))
\]

**effect axiom:** Poss\((a, s)\) \( \Rightarrow \) Changes that result from taking the action.

\[
\forall s \forall x Poss(PlaceOnTable(x, s)) \Rightarrow On(x, Table, Result(PlaceOnTable(x, s))
\]

\[
\forall s \forall y \forall z On(y, z, s) \land (z \neq Table) \Rightarrow \neg On(y, Table, s)
\]

Situation Calculus: Action Sequences

\[
\begin{align*}
Result([], s) &= s \\
Result([a|seq], s) &= Result(seq, Result(a, s))
\end{align*}
\]

We’d like to be able to prove:

\[
\exists seq \ On(A, B, Result(seq, S_0)) \land On(B, C, Result(seq, S_0))
\]

which would produce, for example, the following:

\[
On(A, B, Result([PoT(A), PoT(D), Put(B, C), Put(A, B)], S_0)) \\
\land On(B, C, Result([PoT(A), PoT(D), Put(B, C), Put(A, B)], S_0))
\]

The Frame Problem

Actions don’t specify what happens to objects not involved in the action, but the logic framework requires that information.

\[
\forall s \forall x Poss(PoT(x, s)) \Rightarrow On(x, Table, Result(PoT(x, s))
\]

Frame axioms: Inform the system about preserved relations.

\[
\forall s \forall y \forall z [(On(x, y, s) \land (x \neq z)) \Rightarrow On(x, y, Result(PoT(z, s))]
\]
... and Its Relatives

**representational frame problem:** proliferation of frame axioms.

Solution: use **successor-state axioms**

\[ \text{Action is possible} \Rightarrow (\text{Fluent is true in result state} \iff (\text{Action's effect made it true}) \vee (\text{It was true before and action left it alone})) \]

**inferential frame problem:** have to carry each property through all intervening situations during problem-solving, even if the property remains unchanged throughout.

The Need for Special Purpose Algorithms

So...We have a formalism for expressing goals and plans and we can use resolution theorem proving to find plans.

Problems:
- frame problem
- time to find plan can be exponential
- logical inference is semi-decidable
- resulting plan could have many irrelevant steps

We’ll need to:
- restrict language
- use a special purpose algorithm called a planner

**qualification problem:** difficult, in the real world, to define the circumstances under which a given action is guaranteed to work

**ramification problem:** proliferation of implicit consequences of actions.

The STRIPS Language

**States and Goals:** Conjunctions of positive, function-free literals. No variables.

\[ \text{Have (Milk)} \land \text{Have (Bananas)} \land \text{Have (Drill)} \land \text{At (Home)} \]

Closed world assumption: any conditions that are not mentioned in a state are assumed false.
Actions:

**preconditions:** conjunction of positive, function-free literals that must be true before the operator can be applied.

**effects:** conjunction of function-free literals; add list and delete list.

**STRIPS assumption**
Every literal not mentioned in the effect remains unchanged in the resulting state when the action is executed.
Avoids the representational frame problem.
Solution for the planning problem: an action sequence that, when executed in the initial state, results in a state that satisfies the goal.

**STRIPS Actions**

Move block \( x \) from block \( y \) to block \( z \)
- **preconds:** \( \text{On}(x,y) \land \text{Block}(x) \land \text{Block}(z) \land \text{Clear}(x) \land \text{Clear}(z) \)
- **effects:** Add: \( \text{On}(x,z), \text{Clear}(y) \)  
Delete: \( \text{On}(x,y), \text{Clear}(z) \)

Move block \( x \) from Table to block \( z \)
- **preconds:** \( \text{On}(x,\text{Table}) \land \text{Block}(x) \land \text{Block}(z) \land \text{Clear}(x) \land \text{Clear}(z) \)

<table>
<thead>
<tr>
<th>preconds</th>
<th>effects</th>
</tr>
</thead>
</table>
| Add: \( \text{On}(x,z), \text{Clear}(y) \)  
Delete: \( \text{On}(x,y), \text{Clear}(z) \) |

Move block \( x \) from block \( y \) to Table
- **preconds:** \( \text{On}(x,y) \land \text{Block}(x) \land \text{Block}(y) \land \text{Clear}(x) \)
- **effects:** Add: \( \text{On}(x,\text{Table}), \text{Clear}(y) \)  
Delete: \( \text{On}(x,y) \)
Plan by Searching for a Satisfactory Sequence of Actions

**progression planner** searches forward from the initial situation to the goal situation.

**regression planner** search backwards from the goal state to the initial state.

Heuristics: derive a **relaxed problem**; employ the **subgoal independence** assumption.

Searching Plan Space

Alternative is to search through the space of *plans* rather than the original state space.

Start with simple, incomplete **partial plan**; expand until complete.

**Operators:** add a step, impose an ordering on existing steps, instantiate a previously unbound variable.

**Refinement Operators** take a partial plan and add constraints.

**Modification Operators** are anything that is not a refinement operator; take an incorrect plan and debug it.

Representation for Plans

Goal: *RightShoeOn & LeftShoeOn*

Initial state: λ

Operators:

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconds</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>RightShoe</td>
<td>RightSockOn</td>
<td>RightShoeOn</td>
</tr>
<tr>
<td>RightSock</td>
<td>λ</td>
<td>RightSockOn</td>
</tr>
<tr>
<td>LeftShoe</td>
<td>LeftSockOn</td>
<td>LeftShoeOn</td>
</tr>
<tr>
<td>LeftSock</td>
<td>λ</td>
<td>LeftSockOn</td>
</tr>
</tbody>
</table>

Partial Plans

Partial Plan: RightShoe LeftShoe

Principle of **Least Commitment** says to only make choices about things that you currently care about.

**Partial order planner** – can represent plans in which some steps are ordered and others are not.

**Total order planner** considers a plan a simple list of steps.

A **linearization of P** is a totally ordered plan that is derived from a plan P by adding ordering constraints.
Defer Variable Binding

Planners must commit to bindings for variables

Example: Goal: Have(Milk) Action: Buy(item,store)

**Principle of Least Commitment:** Only make choices about things that you care about, leaving other details to be worked out later.

Buy(Milk,K-MART) versus Buy(Milk,store)

**Fully instantiated plan:** every variable is bound to a constant.

---

**Definition of a Plan**

- A set of plan steps (actions).
- A set of step ordering constraints of the form $S_i \prec S_j$
- A set of variable binding constraints
- A set of causal links, written as $S_i \xrightarrow{c} S_j$

---

**Initial Plan for Shoes and Socks**

Initial plan: $Start \prec Finish$

(a) Start

Initial State

Goal State

Finish

(b) Start

LeftShoeOn, RightShoeOn

Finish

---

**Partial Plan for Shoes and Socks**

Partial Order Plan:

Start

Left Sock, Right Sock

LeftShoeOn, RightShoeOn

Finish

Total Order Plans:

Start

Right Sock, Left Sock

RightShoeOn, LeftShoeOn

Finish

---

---
Planner Output

A solution is a complete, consistent plan.

A **complete plan**: every precondition of every step is achieved by some other step.

A **consistent plan**: there are no contradictions in the ordering or binding constraints. Contradiction occurs when both $S_i \prec S_j$ and $S_j \prec S_i$.

### POP Example: STRIPS Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>PreCond</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go(there)</td>
<td>At(here)</td>
<td>$At(there) \land \neg At(here)$</td>
</tr>
<tr>
<td>Buy($x$)</td>
<td>$At(store) \land Sells(store, x)$</td>
<td>$Have(x)$</td>
</tr>
</tbody>
</table>

### POP Example: Initial Plan

- **Initial Plan**
  - At(Home)  Sells(SM,Banana)  Sells(SM,Milk)  Sells(HWS,Drill)
  - Have(Drill)  Have(Milk)  Have(Banana)  At(Home)
  - Start

### A Partial Plan I

- **Partial Plan I**
  - At(SM), Sells(SM,Milk)  At(SM), Sells(SM,Drill)  At(SM), Sells(SM,Banana)
  - Have(Drill), Have(Milk), Have(Banana), At(Home)
  - Start
A Partial Plan II

At(SM), Sells(SM,Bananas) At(SM), Sells(SM,Milk) At(HWS), Sells(HWS,Drill)

Have(Drill), Have(Milk), Have(Bananas), At(Home)

Finish

Go(HWS)

Buy(Drill)

Go(SM)

Buy(Milk)

Buy(Bananas)

A Partial Plan III

At(Home), Sells(Home,Drill) At(SM), Sells(SM,Milk) At(SM), Sells(SM,Bananas)

Have(Drill), Have(Milk), Have(Bananas), At(Home)

Finish

Go(HWS)

Buy(Drill)

Go(SM)

Buy(Milk)

Buy(Bananas)

Protecting Causal Links

(a) (b) (c)

A Partial Plan III

At(Home), Sells(Home,Drill) At(SM), Sells(SM,Milk) At(SM), Sells(SM,Bananas)

Have(Drill), Have(Milk), Have(Bananas), At(Home)

Finish

Go(Home)
Achieving At(Home)

Candidate link Threats
At(x) to initial state \( \text{Go(HWS), Go(SM)} \)
At(x) to \( \text{Go(HWS)} \) \( \text{Go(SM)} \)
At(x) to \( \text{Go(SM)} \) \( \text{At(SM) preconds of Buy(Milk), Buy(Bananas)} \)

**Solution:** Link At(x) to Go(SM), but order Go(Home) to come after Buy(Bananas) and Buy(Milk).

---

**Slide CS472 – Planning 37**

**Slide CS472 – Planning 38**

---

**function** \( \text{POP(initial, goal, operators)} \) **returns** plan

\[
\begin{align*}
\text{plan} & \leftarrow \text{MAKE-MINIMAL-PLAN(initial, goal)} \\
\text{loop} & \text{do} \\
\text{if} \ & \text{SOLUTION?}(\text{plan}) \text{ then return } \text{plan} \\
\text{Sneed, c} & \leftarrow \text{SELECT-SUBGOAL}(\text{plan}) \\
\text{CHOOSE-OPERATOR}(\text{plan}, \text{operators}, \text{Sneed, c}) \\
\text{RESOLVE-THREATS}(\text{plan}) \\
\text{end}
\end{align*}
\]

**function** \( \text{SELECT-SUBGOAL}(\text{plan}) \) **returns** \( \text{Sneed, c} \)

pick a plan step \( \text{Sneed} \) from \( \text{STEPS(plan)} \)
with a precondition \( c \) that has not been achieved

**procedure** \( \text{CHOOSE-OPERATOR}(\text{plan}, \text{operators, Sneed, c}) \)

choose a step \( \text{Sadd} \) from \( \text{operators} \) or \( \text{STEPS(plan)} \) that has \( c \) as an effect
if there is no such step then fail
add the causal link \( \text{Sneed} \leftarrow \text{Sadd} \) to \( \text{LINKS(plan)} \)
add the ordering constraint \( \text{Sadd} \leftarrow \text{Sneed} \) to \( \text{ORDERINGS(plan)} \)
if \( \text{Sadd} \) is a newly added step from \( \text{operators} \) then
add \( \text{Sadd} \) to \( \text{STEPS(plan)} \)
add \( \text{Start} \leftarrow \text{Sadd} \) \( /; /! \) \( \text{Finish} \) to \( \text{ORDERINGS(plan)} \)

**procedure** \( \text{RESOLVE-THREATS(plan)} \)

for each \( \text{Sthreat} \) that threatens a link \( \text{Sneed} \leftarrow \text{Sadd} \) in \( \text{LINKS(plan)} \) do
choose either
Promotion: Add \( \text{Sneed} \leftarrow \text{Sadd} \) to \( \text{LINKS(plan)} \)
Deposition: Add \( \text{Sadd} \leftarrow \text{Sneed} \) to \( \text{ORDERINGS(plan)} \)
if \( \text{not CONSISTENT(plan)} \) then fail
end

**Strengths of Partial-Order Planning Algorithms**

- Takes a huge state space problem and solves in only a few steps.
- Least commitment strategy means that search only occurs in places where sub-plans interact.
- Causal links allow planner to recognize when to abandon a doomed plan without wasting time exploring irrelevant parts of the plan.
Practical Planners

STRIPS approach is insufficient for many practical planning problems. Can’t express:

resources: Operators should incorporate resource consumption and generation. Planners have to handle constraints on resources efficiently.

time: Real-world planners need a better model of time.

hierarchical plans: Need the ability to specify plans at varying levels of detail.

Also need to incorporate heuristics for guiding search.

Planning Graphs

- Data structure (graphs) that represents plans, and can be efficiently constructed, and that allows for better heuristic estimates.

- Graphplan: algorithm that processes the planning graph, using backward search, to extract a plan.

- SATPlan: algorithm that translates a planning problem into propositional axioms and applies a CSP algorithm to find a valid plan.

Take CS672 to learn more!!!

Hierarchical Planning

Spacecraft Assembly, integration and verification (AIV)

- OPTIMUM-AIV used by the European Space Agency to AIV spacecraft.

- Generates plans and monitors their execution – ability to re-plan is the principle objective.

- Uses O-Plan architecture – like partial-order planner, but can represent time, resources and hierarchical plans. Accepts heuristics for guiding search and records its reasons for each choice.
Scheduling for Space Missions

- Planners have been used by the ground teams for the Hubble space telescope and for the Voyager, Uosat-II and ERS-1.

- Goal: coordinate the observational equipment, signal transmitters and altitude and velocity-control mechanism in order to maximize the value of the information gained from observations while obeying resource constraints on time and energy.