Sussman Anomaly in the block world

"Sussman anomaly" problem

Start State

\[
\begin{align*}
\text{Clear}(x) & \land \text{On}(x, z) \land \text{Clear}(y) \\
\text{PutOn}(x, y) & \\
\neg \text{On}(x, z) & \land \neg \text{Clear}(y) \\
\text{Clear}(z) & \land \text{On}(x, y)
\end{align*}
\]

Goal State

\[
\begin{align*}
\text{Clear}(x) & \land \text{On}(x, z) \\
\text{PutOnTable}(x) & \\
\neg \text{On}(x, z) & \land \text{Clear}(z) \land \text{On}(x, \text{Table})
\end{align*}
\]

+ several inequality constraints
Sussman Anomaly

- The Sussman Anomaly shows the limitations of non-interleaved planning methods
- Before this was described, people used to do planning by considering different subgoals in SEQUENCE
- The Anomaly will show that naively pursuing one subgoal X after you satisfy the other subgoal Y may not work because steps required to accomplish X might undo things subgoal Y
Sussman Anomaly

- Final state requires On(A,B) and On(B, C)
- Top diagram tries to focus on subgoal: On(B, C) -- Now trying to put A on top of B cannot be done without undoing On(B, C)
- Bottom diagram tries to focus on subgoal: On(A, B) first; but now trying to put B on top of C would cause On(A,B) to be undone!
Anomaly Illustrates the Need for Interleaved Plans

\[ \text{On}(C, A) \text{ On}(A, \text{Table}) \text{ Cl}(B) \text{ On}(B, \text{Table}) \text{ Cl}(C) \]

\[ \text{On}(A, B) \text{ On}(B, C) \]

\[ \text{START} \hspace{2cm} \text{FINISH} \]
Example: continued

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

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

PutOn(B,C)

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

FINISH
Need to Re-Order Plan Steps Dynamically

START

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

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

$On(A,B)$

PutOn$(A,B)$

PutOn$(B,C)$

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

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

FINISH

PutOn$(A,B)$ clobbers $Cl(B)$ => order after PutOn$(B,C)$
Example (cont.)
Conclusion from the Blocks Example

- Problem can be solved, BUT not by trying to apply ALL operators to achieve a single goal at a time sequentially – satisfying one goal seems to clobber earlier achieved goals.
- The issue: we are forcing an order on operators when they do not need to be mutually ordered.
- We need an approach that allows INTERLEAVING of steps for multiple goals
- This observation motivates the next planning approach: PARTIAL ORDER PLANNING - to be covered next class...
Partial-order planning

- Progression and regression planning are *totally ordered plan search* forms.
  - They cannot take advantage of problem decomposition.
    - Decisions must be made on how to sequence actions on all the subproblems
- Least commitment strategy:
  - Delay choice during search
Shoe example

Goal(RightShoeOn \land LeftShoeOn)
Init()
Action(RightShoe,  \quad \text{PRECOND}: \text{RightSockOn}
   \quad \text{EFFECT}: \text{RightShoeOn})
Action(RightSock,  \quad \text{PRECOND}:
   \quad \text{EFFECT}: \text{RightSockOn})
Action(LeftShoe,  \quad \text{PRECOND}: \text{LeftSockOn}
   \quad \text{EFFECT}: \text{LeftShoeOn})
Action(LeftSock,  \quad \text{PRECOND}:
   \quad \text{EFFECT}: \text{LeftSockOn})

Planner: combine two action sequences (1)leftsock, leftshoe (2)rightsock, rightshoe
Partial-order planning

- Any planning algorithm that can place two actions into a plan without commitment about which comes first is a Partial Order Plan.

![Diagram of partial-order planning and total order plans for putting on socks and shoes.](image)
Partial Order Planning as a search problem

- States are (mostly unfinished) plans.
  - The empty plan contains only start and finish actions.
- Each plan has 4 components:
  - A set of actions (steps of the plan)
  - A set of ordering constraints: \( A < B \)
    - Cycles represent contradictions.
  - A set of causal links \( A \xrightarrow{p} B \)
    - The plan may not be extended by adding a new action \( C \) that conflicts with the causal link. (if the effect of \( C \) is \( \neg p \) and if \( C \) could come after \( A \) and before \( B \))
  - A set of open preconditions.
    - If precondition is not achieved by action in the plan.
POP as a search problem

- A plan is *consistent* iff there are no cycles in the ordering constraints and no conflicts with the causal links.
- A consistent plan with no open preconditions is a *solution*.
- A partial order plan is executed by repeatedly choosing *any* of the possible next actions.
  - This flexibility is a benefit in non-cooperative environments.
Solving POP

- Assume propositional planning problems:
  - The initial plan contains \textit{Start} and \textit{Finish}, the ordering constraint \textit{Start} < \textit{Finish}, no causal links, all the preconditions in \textit{Finish} are open.
  - Successor function:
    - picks one open precondition $p$ on an action $B$ and
    - generates a successor plan for every possible consistent way of choosing action $A$ that achieves $p$.
  - Test goal
Enforcing consistency

- When generating successor plan:
  - The causal link A--p->B and the ordering constraint A < B is added to the plan.
    - If A is new also add start < A and A < B to the plan
  - Resolve conflicts between new causal link and all existing actions
  - Resolve conflicts between action A (if new) and all existing causal links.
Process summary

- Operators on partial plans
  - Add link from existing plan to open precondition.
  - Add a step to fulfill an open condition.
  - Order one step w.r.t another to remove possible conflicts
- Gradually move from incomplete/vague plans to complete/correct plans
- Backtrack if an open condition is unachievable or if a conflict is unresolvable.
Example: Spare tire problem

\[\text{Init}(At(Flat, Axle) \land At(Spare, trunk))\]

\[\text{Goal}(At(Spare, Axle))\]

\[\text{Action}(\text{Remove}(Spare, Trunk))\]

\[\text{PRECOND: } At(Spare, Trunk)\]

\[\text{EFFECT: } \neg At(Spare, Trunk) \land At(Spare, Ground))\]

\[\text{Action}(\text{Remove}(Flat, Axle))\]

\[\text{PRECOND: } At(Flat, Axle)\]

\[\text{EFFECT: } \neg At(Flat, Axle) \land At(Flat, Ground))\]

\[\text{Action}(\text{PutOn}(Spare, Axle))\]

\[\text{PRECOND: } At(Spare, Ground) \land \neg At(Flat, Axle)\]

\[\text{EFFECT: } At(Spare, Axle) \land \neg At(Spare, Ground)\]

\[\text{Action}(\text{LeaveOvernight})\]

\[\text{PRECOND: }\]

\[\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, trunk) \land \neg At(Flat, Ground) \land \neg At(Flat, Axle)\]
Solving the problem

- Initial plan: Start with EFFECTS and Finish with PRECOND.
Solving the problem

- Initial plan: Start with EFFECTS and Finish with PRECOND.
- Pick an open precondition: \textit{At(Spare, Axle)}
- Only \textit{PutOn(Spare, Axle)} is applicable
- Add causal link: \textit{PutOn(Spare, Axle)} \rightarrow \textit{At(Spare, Axle)} \rightarrow \textit{Finish}
- Add constraint: \textit{PutOn(Spare, Axle)} < \textit{Finish}
Solving the problem

- Pick an open precondition: \( At(Spare, \text{Ground}) \)
- Only \( Remove(Spare, \text{Trunk}) \) is applicable
- Add causal link: \( Remove(Spare, \text{Trunk}) \xrightarrow{At(Spare,\text{Ground})} PutOn(Spare, \text{Axle}) \)
- Add constraint: \( Remove(Spare, \text{Trunk}) < PutOn(Spare, \text{Axle}) \)
Solving the problem

- Pick an open precondition: $\neg At(Flat, Axle)$
- $LeaveOverNight$ is applicable
- conflict: $\text{Remove}(Spare, Trunk) \rightarrow \text{PutOn}(Spare, Axle) \\ At(Spare, Ground)$

- Because $LeaveOverNight$ also makes $\neg At(Spare, Ground)$
- To resolve, add constraint: $LeaveOverNight < Remove(Spare, Trunk)$
Solving the problem

- Pick an open precondition: \textit{At(Spare, Trunk)}
- Only \textit{Start} is applicable
- Add causal link: \textit{Start} \xrightarrow{\textit{At(Spare,Trunk)}} \textit{Remove(Spare,Trunk)}
- Conflict: of causal link with effect \(\rightarrow \textit{At(Spare,Trunk)}\) in \textit{LeaveOverNight}
  - \textit{No re-ordering solution possible.}
- Backtrack to a prior move since there is no way to fix this
Solving the problem

- Backtracking step: Remove *LeaveOverNight* and its causal links
- Now try *Remove(Flat, Axle)* as a way to satisfy \( \neg At(Flat, Axle) \)
- That one works... and the partial plan can be completed as above
Some details …

- What happens when a first-order representation that includes variables is used?
  - Complicates the process of detecting and resolving conflicts.
  - Can be resolved by introducing inequality constraints.
- CSP’s most-constrained-variable constraint can be used for planning algorithms to select a PRECOND.
Planning graphs

- Used to achieve better heuristic estimates.
  - A solution can also directly extracted using GRAPHPLAN.
- Consists of a sequence of levels that correspond to time steps in the plan.
  - Level 0 is the initial state.
  - Each level consists of a set of literals and a set of actions.
    - *Literals* = all those that *could* be true at that time step, depending upon the actions executed at the preceding time step.
    - *Actions* = all those actions that *could* have their preconditions satisfied at that time step, depending on which of the literals actually hold.
Planning graphs

• “Could”?  
  • Records only a restricted subset of possible negative interactions among actions.

• They work only for propositional problems.

• Example:

  Init(Have(Cake))
  
  Goal(Have(Cake) \& Eaten(Cake))
  
  Action(Eat(Cake), PRECOND: Have(Cake))
  
    EFFECT: \neg Have(Cake) \& Eaten(Cake))
  
  Action(Bake(Cake), PRECOND: \neg Have(Cake))
  
    EFFECT: Have(Cake))
Cake example

- Start at level S0 and determine action level A0 and next level S1.
  - A0 >> all actions whose preconditions are satisfied in the previous level.
  - Connect precondition and effect of actions S0 --> S1
  - Inaction is represented by persistence actions.
- Level A0 contains the actions that could occur
  - Conflicts between actions are represented by *mutex* links
Cake example

- Level S1 contains all literals that could result from picking any subset of actions in A0
  - Conflicts between literals that cannot occur together are represented by mutex links.
  - S1 defines multiple states and the mutex links are the constraints that define this set of states.
- Continue until two consecutive levels are identical: leveled off
  - Or contain the same amount of literals (explanation follows later)
Cake example

- A mutex relation holds between **two actions** when:
  - *Inconsistent effects*: one action negates the effect of another (Have(Cake) and Eat(Cake) for example)
  - *Interference*: one of the effects of one action is the negation of a precondition of the other. Eat(Cake) negates the precondition of Have(Cake) persistence and therefore interferes with it
  - *Competing needs*: one of the preconditions of one action is mutually exclusive with the precondition of the other. For example, Bake(Cake) competes with Eat(Cake) on the Have(Cake) pre condition.

- A mutex relation holds between **two literals** when (**inconsistent support**):
  - If one is the negation of the other OR
  - if each possible action pair that could achieve the literals is mutex.
Plan Graphs and heuristic estimation

- PG’s provide information about the problem
  - A literal that does not appear in the final level of the graph cannot be achieved by any plan.
    - Useful for backward search (cost = inf).
  - Level of appearance can be used as cost estimate of achieving any goal literals = \( \text{level cost} \).
- Small problem: several actions can occur
  - Restrict to one action using serial PG (add mutex links between every pair of actions, except persistence actions).
  - Max-level, sum-level and set-level heuristics.

PG is a relaxed problem.
The GRAPHPLAN Algorithm

- How to extract a solution directly from the PG

```python
function GRAPHPLAN(problem) return solution or failure
  graph ← INITIAL-PLANNING-GRAPH(problem)
  goals ← GOALS[problem]
  loop do
    if goals all non-mutex in last level of graph then do
      solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
      if solution ≠ failure then return solution
    else if NO-SOLUTION-POSSIBLE(graph) then return failure
    graph ← EXPAND-GRAPH(graph, problem)
```

CS 460, Session 20
GRAPHPLAN example

- Initially the plan consist of 5 literals from the initial state and the CWA literals (S0).
- Add actions whose preconditions are satisfied by EXPAND-GRAPH (A0)
- Also add persistence actions and mutex relations.
- Add the effects at level S1
- Repeat until goal state appears in some level
GRAPHPLAN example

- EXPAND-GRAPH also looks for mutex relations
  - Inconsistent effects
    - E.g. Remove(Spare, Trunk) and LeaveOverNight
  - Interference
    - E.g. Remove(Flat, Axle) and LeaveOverNight
  - Competing needs
    - E.g. PutOn(Spare, Axle) and Remove(Flat, Axle)
  - Inconsistent support
    - E.g. in S2, At(Spare, Axle) and At(Flat, Axle)
GRAPHPLAN example

- In S2, the goal literal exists and is not mutex with any other
  - Solution might exist and EXTRACT-SOLUTION will try to find it
- EXTRACT-SOLUTION can use Boolean CSP to solve the problem or a search process:
  - Initial state = last level of PG and goal goals of planning problem
  - Actions = select any set of non-conflicting actions that cover the goals in the state
  - Goal = reach level S0 such that all goals are satisfied
  - Cost = 1 for each action.
Termination? YES

PG are monotonically increasing or decreasing:
- Literals increase monotonically
- Actions increase monotonically
- Mutexes decrease monotonically

Because of these properties and because there is a finite number of actions and literals, every PG will eventually level off!
Analysis of planning approach

- Planning is an area of great interest within AI
  - Search for solution
  - Constructively prove a existence of solution
- Biggest problem is the combinatorial explosion in states.
- Efficient methods are under research
  - E.g. divide-and-conquer