Partial Order Planning
Good article for this topic:

- An Introduction to Least Commitment Planning  AI Magazine
  - Dan Weld
So far we have considered planning as search in state space
- **Forward** - build a plan in the same order that it is executed
- **Backward** - build a plan in the reverse order of its execution
Potential problem:
Spending lots of time on trying the same set of actions in different orderings before realizing that there is no solution (with this set)

Key observation: In these algorithms, when we choose what to do, we also choose when to do it
In 1974, Earl Sacerdoti built a planner, called NOAH, that considered planning as search through plan space.

- Search states (nodes) = partially specified plans
- Transitions (edges) = plan refinement operations
- Initial state = null plan
- Goal states = valid plans for the problems
State Space vs. Plan Space

Search through plan space: what is a plan?

- **Answer I:** Totally ordered sequences of actions  
  - But then search through state space is **isomorphic** to search through plan space!
  - So what is the point of introducing search through "plan space"?

- **Answer II:** Opens the road to more interesting plan representations and more interesting operators on plans, in particular, **partially ordered** sequence of actions
Least Commitment Planning

- Think how you might solve a planning problem of going for a vacation in Italy
  - Need to purchase plane tickets
  - Need to buy the "Lonely Planet" guide to Italy

BUT there is no need to decide (yet) which purchase should be done first

Least Commitment Planning

- Represent plans in a flexible way that enables deferring decisions
- At the planning phase, only the essential ordering decisions are recorded
Given a Strips task $\prod = (P, A, I, G)$ we search through a space of hypothetical **partial-order plans**.

A plan (= search node) is a triplet: $\langle A, O, L \rangle$ in which:

- $A$ is a set of **actions** from $A$, possibly with (labeled) repetitions.
- $O$ is a set of **ordering constraints** over $A$.
- $L$ is a set of **causal links** (a bit later).

Example: $A = \{a_1, a_2, a_3\}$, $O = \{a_1 < a_3, a_2 < a_3\}$

Planner must ensure the **consistency** of $O$. 
A key aspect of least commitment planning: keep track of past decisions and the reasons for those decisions

- If you purchase plane tickets early to board the plane, make sure they’re with you when you get to the airport.
- If another goal causes you to drop the tickets (e.g., having your hands free to open the taxi door), then you should be sure to pick them up again.
- A good way to reason about and ensure non-interference between different actions introduced into the plan is to record dependencies between actions explicitly.
- Causal links $a_p \rightarrow q a_c$ records our decision to use $a_p$ to produce the precondition $q$ of $a_c$. 
Threats

- Causal links are used to detect when a newly introduced action interferes with past decisions.
- Such an action is called a threat.

- Suppose that
  - $a_p \xrightarrow{q} a_c$ is a causal link in $L$ (of some plan $\langle A, O, L \rangle$)
  - $a_t$ is yet another action in $A$

- We say that at threatens $a_p \xrightarrow{q} a_c$ if
  - $O \cup \{a_p < a_t < a_c\}$ is consistent, and
  - $q \in \text{del}(a_t)$
Eliminating Threats

- When a plan contains a threat, then it is possible that the plan would not work as anticipated.
  - Which means what?
- Solution: identify threats and take evasive countermeasures
  - promotion by \( O \cup = \{ a_t > a_c \} \)
  - demotion by \( O \cup = \{ a_t < a_p \} \)
  - ...

...
Planning Problems as Null Plans

Uniformity is the key to simplicity

- Can use the same structure to represent both the planning problem and complete plans
- Planning problem as a null plan \( \langle A, O, L \rangle \) where
  - \( A = \{ a_0, a_\infty \} \), \( O = \{ a_0 < a_\infty \} \), \( L = \{ \} \)
  - \( pre(a_0) = \{ \}, del(a_0) = \{ \}, add(a_0) = I \)
  - \( pre(a_\infty) = G, del(a_\infty) = \{ \}, add(a_\infty) = \{ \} \)

*start*

(on c a) (clear b) (clear c) (on a table) (on b table)

(on a b) (on b c)

*end*
The POP Algorithm
Schematic description

Regressive algorithm that searches plan space

- Starts with the null plan
- Makes non-deterministic plan refinement choices until
  - all preconditions of all actions in the plan have been supported by causal links, and
  - all threats against any causal link have been removed
The POP Algorithm

Input and Output

- Recursive calls to POP with \( \text{POP}(\langle A, O, L \rangle, \text{agenda}, A) \) where
  - \( \langle A, O, L \rangle \) is a plan structure.
  - agenda is a list of "open goals" that need to be supported by causal links.
  - \( A \) is the action set of our Strips problem.

- Initial call is with
  - null plan \( \langle \{a_0, a_\infty\}, \{a_0 < a_\infty\}, \{\} \rangle \)
  - agenda = \( \{(g, a_\infty)|g \in \text{pre}(a_\infty) \equiv G\} \)

If \( \langle A, O, L \rangle \) is outputted by POP, then any total ordering of actions \( A \) consistent with \( O \) is a valid plan for our problem.
The POP Algorithm

\[ \text{POP}(\langle A, O, L \rangle, \text{agenda}, A) \]

- **Termination:**
  \[ \text{if agenda} = \emptyset \text{ then return } \langle A, O, L \rangle \]

- **Goal selection:**
  \[ \text{choose } (q, a_{\text{need}}) \in \text{agenda} \]

- **Action selection...**
The POP Algorithm

- **Action Selection:**
  - choose action $a_{add}$ (either from $A$, or from $A$) such that
    - $q \in \text{add}(a_{add})$, and
    - $O \cup \{a_{add} < a_{need}\}$ is consistent
  - if no such action then return FALSE
  - otherwise
    - $L \cup = \{a_{add} \rightarrow q, a_{need}\}$ and $O \cup = \{a_{add} < a_{need}\}$
    - if $a_{add}$ is a new action instance then
      - $A \cup = \{a_{add}\}$, and $O \cup = \{a_0 < a_{add} < a_{\infty}\}$

- **Update goal set:**
  - agenda $\setminus = \{(q, a_{need})\}$
  - if $a_{add}$ was a new action instance then
    - agenda $\cup = \{(r, a_{add})| r \in \text{pre}(a_{add})\}$
The POP Algorithm

POP(⟨A, O, L⟩, agenda, A)

- **Termination:** if agenda = then return ⟨A, O, L⟩
- **Goal selection:** choose (q, a_{need}) ∈ agenda
- **Action selection:** choose and process a_{add}...
- **Update goal set:** add preconditions of a_{add} to the agenda...

- **Causal link protection:**
  - foreach causal link \{a_p \rightarrow^q a_c\} ∈ L, and a_t that threatens it
  - choose either O ∪ = a_t > a_c, or O ∪ = \{a_t < a_p\}
  - if neither constraint is consistent, then return FALSE
- **Recursive invocation:** POP(⟨A, O, L⟩, agenda, A)
The POP Algorithm

In one slide ...

\[ \text{POP}(\langle A, O, L \rangle, \text{agenda, A}) \]

- **Termination:**
  \[ \text{if agenda} = \emptyset \text{ then return } \langle A, O, L \rangle \]

- **Goal selection:**
  choose \((q, a_{\text{need}}) \in \text{agenda}\)

- **Action selection:**
  
  - **choose** action \(a_{\text{add}}\) (either from \(A\), or from \(A\)) such that
    
    - \(q \in \text{add}(a_{\text{add}})\), and
    
    - \(O \cup \{a_{\text{add}} < a_{\text{need}}\}\) is consistent

  \[ \text{if no such action then return FALSE} \]

  \[ \text{otherwise} \]

  - \(L \cup = \{a_{\text{add}} \xrightarrow{q} a_{\text{need}}\}\) and \(O \cup = \{a_{\text{add}} < a_{\text{need}}\}\)

  \[ \text{if } a_{\text{add}} \text{ is a new action instance then } A \cup = \{a_{\text{add}}\}, \text{ and } O \cup = \{a_0 < a_{\text{add}} < a_\infty\}\]
The POP Algorithm

In one slide ...

- **Update goal set:**
  - agenda \( \setminus \) = \( \{(q, a_{\text{need}})\} \)
  - if \( a_{\text{add}} \) was a new action instance then agenda
    \[ \bigcup = \{(r, a_{\text{add}})\mid r \in \text{pre}(a_{\text{add}})\} \]

- **Causal link protection:**
  - foreach causal link \( \{a_p \xrightarrow{q} a_c\} \in L \), and at that threatens it
    - **choose** either \( O \cup = \{a_t > a_c\} \), or \( O \cup = \{a_t < a_p\} \)
    - if neither constraint is consistent, then return FALSE

- **Recursive invocation:** POP(\( \langle A, O, L \rangle, \text{agenda}, A \))
Choice Points

Three choice points
- Goal selection
- Action selection
- Causal link protection

How crucial these choices are?
- Affect soundness?
- Affect completeness?
- Affect efficiency?
Example - Step 1

Initial call to POP with
- Null Plan (see the right figure)
- agenda = \{ (onAB, a_\infty), (onBC, a_\infty) \}

First choice is goal selection
- Affects efficiency, but not completeness!
Example - Step 2

Suppose \((onBC, a_{\infty})\) is selected (i.e., \(a_{\text{need}} = a_{\infty}\))

- Need to choose an action \(a_{\text{add}}\) that will provide \(onBC\)
  - This is a real non-deterministic choice!

Suppose that an oracle suggests making \(a_{\text{add}}\) a new instance of the action move-B-from-Table-to-C

- a causal link \(a_{\text{add}} \xrightarrow{onBC} a_{\infty}\) is added to \(L\)
- agenda is properly updated (how exactly?)
- no threats to resolve . . . recursive call
Example - Step 2

Suppose that an oracle suggests making $a_{\text{add}}$ a new instance of the action move-B-from-Table-to-C

- a causal link $a_{\text{add}} \overset{\text{onBC}}{\rightarrow} a_{\infty}$ is added to L
- agenda is properly updated (how exactly?)
- no threats to resolve . . . recursive call

```
*start*
(on c a) (clear b) (clear c) (on a table) (on b table)

(clear b) (clear c) (on b table)

(move b from table to c)

(clear table) \neg (on b table) \neg (clear c) (on b c)

(on a b) (on b c)

*end*
```
Example - Step 3

Suppose \((\text{clearB, move-B-from-Table-to-C})\) is selected

- Oracle suggests to reuse an **existing** action instance \(a_0\)
  - add a causal link \(a_0 \rightarrow \text{move-B-from-Table-to-C}\)
  - agenda is properly updated (how exactly?)
  - no threats to resolve . . . recursive call

```
*start*

(on c a) (clear b) (clear c) (on a table) (on b table)

\(\leftarrow\) (clear b) (clear c) (on b table)

(move b from table to c)

(clear table) \neg(on b table) \neg(clear c) (on b c)

(on a b) (on b c)

*end*
```
Example - Step 4a

- Suppose \((\text{onAB, } a_{\infty})\) is selected
- Oracle suggests making \(a_{\text{add}}\) be a new instance of the action \(\text{move-A-from-Table-to-B}\), and we do that ... 
- ... BUT this time we have a threat!
  - \(\text{move-A-from-Table-to-B}\) and \(\text{move-B-from-Table-to-C}\) have no constraints on their relative ordering
  - \(\text{move-A-from-Table-to-B}\) deletes \(\text{clearB}\) that is required by \(\text{move-B-from-Table-to-C}\)
Example - Step 4b

Try to **protect** the causal link $a_0^{clearB} \rightarrow$ move-B-from-Table-to-C

- In general, there are two options promotion and demotion and this is a true non-deterministic choice!
- In our example, demotion is inconsistent (why?), but promotion is OK

```
*start*

(on c a) (clear b) (clear c) (on a table) (on b table)

(move a from table to b)

(clear b) (clear a) (on a table)

(clear b) (clear c) (on b table)

(move b from table to c)

(clear table) \neg( on b table) \neg( clear c) (on b c)

(on a b) (clear table) \neg( on a table) \neg( clear b)

(on a b) (on b c)

*end*
```
Example - Next steps

What is now on the agenda? ... in A? ... in L? ... In O?

Next steps follow the same lines of reasoning ...
Example - Next steps

Eventually POP returns

*start*

(on c a) (clear b) (clear c) (on a table) (on b table)

(on e a) (clear c)

(move c from a to table)

(clear a) (on c table) (on c a)

(clear b) (clear a) (on a table)

(move a from table to b)

(clear b) (clear c) (on b table)

(move b from table to c)

(on a b) (clear table) ¬(on a table) ¬(clear b)

(clear table) ¬(on b table) ¬(clear c) (on b c)

(on a b) (on b c)

*end*

Is it a correct partial order plan?
Advantages

- Natural extension to planning with **partially instantiated actions**
  - ... add action instance `move(A, x, B)`
  - ... postpone unifying `?x` with a concrete object until necessary
- Natural extensions to more complex **action formalisms**
  - ... action durations
  - ... delayed effects
  - ...
- Least commitment may lead to shorter search times
  - Mainly due to smaller **branching factor**
- Another way of viewing it: constraint-based planning
Disadvantages

- Significantly more complex algorithm
  - ... higher per-node cost
- Hard to determine what is true in a state
  - ... harder to devise informed heuristics (for all three types of choices)
  - ... how to prune infinitely long paths??


3. ... for many more, see, e.g., bibliography in (1)