Transformational Partial Order Planning

Håkan L. Younes

Computer Science Department
Carnegie Mellon University

Outline

- Background
- Threat Resolution in Refinement Planning
- Transformational Threat Resolution
- Soundness and Completeness
- Search Control Heuristics
- Conclusions

Refinement Planning

- Flaws are corrected by adding elements (causal links, steps, ordering constraints, and binding constraints) to an incomplete plan.
- If a dead-end is reached, backtracking is used to retreat to a node where a different choice can be made.

Transformational Planning

A transformational planner is not restricted to backtracking as only option for retracting refinements.

Related Work

Gordius (Simmons, 1987):

- Transformational operators to correct flaws in a plan.
- Not a least commitment planner.

SPA (Hanks & Weld, 1995):

- Based on SNLP (partial order planner).
- Transformational operators limited to retracting most recent refinement (only used when initial plan is retrieved from a plan library).

Threat Resolution in Refinement Planning

A causal link $S_i \xrightarrow{q} S_j$ is threatened by a step S_k if it is possible that S_k can be ordered between S_i and S_j, and an effect e of S_k can be unified with $\overline{q}$. A threat can be resolved in three ways:

- *Promotion*: Order S_k after S_j.
- *Demotion*: Order S_k before S_i.
- *Separation*: Add variable bindings that make e non-unifiable with $\overline{q}$.

Transformational Threat Resolution

A threat against a causal link $S \xrightarrow{q} S_j$ posed by S_k can be resolved by linking the condition q to a different effect. To do this we

- Remove the causal link $S \xrightarrow{q} S_j$.
- Create new link $S' \xrightarrow{q} S_j$, where S' could be an existing step or a completely new step.

Logging Reasons for Refinements

Using reasons as an "undo log":

- Adding step S_i: Variable bindings associated with the new step's action schema are added with reason [ADD-STEP S_i]

- Adding link $S_i \xrightarrow{q} S_j$: S_i, and any ordering and binding constraint added gets reason [ESTABLISH $S_i \xrightarrow{q} S_j$] attached.

- Threat resolution: Any ordering or binding constraint added to resolve a threat against $S_i \xrightarrow{q} S_j$ by S_k gets reason [PROTECT $\langle S_i \xrightarrow{q} S_j, S_k \rangle$] attached.

Element without reason is exposed.

Causal Link Removal

When removing a causal link $S_i \xrightarrow{q} S_j$ we must:

- Remove all flaws involving link.
- Add open condition $\xrightarrow{q} S_j$ to flaws.
- Remove all reasons involving link.

Step Removal

When removing a step S_k we must:

- Remove all links involving step.
- Remove all flaws involving step.
- Remove all reasons involving step.

Removal of Exposed Elements

To keep the plan clean, we remove all exposed elements—i.e. all the steps, links, orderings, and bindings we no longer have a reason to keep.

Consequences of Constraint Removal

If a binding or ordering constraint is removed, this could have non-local consequences, and can introduce new flaws.

Hard to determine new flaws. May have to test all remaining links against all remaining steps!

Soundness

All solutions found must be legal plans!

Make sure no threats or open conditions are overlooked when transformational operators are applied.

Completeness

If a solution exists, it should be found!

Make sure cycles are avoided. A plan should not be added to the search horizon when an equivalent plan has been seen along the search path already.

Otherwise, we rely on the completeness of the underlying refinement planner.

Plan Equivalence

Two plans P_1 and P_2 are equivalent if

1. there is a bijective mapping M from the steps in P_1 to the steps in P_2, and given M
   (a) there is a one to one correspondence between links in P_1 and P_2, and
   (b) the transitive closures of the ordering constraints match, and
   (c) the codesignation and non-codesignation sets for variables coincide.

Search Control Heuristics

Crucial with good search control heuristics because transformational operators are expensive and branching factor is increased!

When to invoke transformational operator?

- At any time.
- Only when refinement is not possible.

Can the cost of a transformational operator be estimated before it is invoked?

Conclusions

- Strongly typed languages are good!
- Transformational operators introduce new search opportunities, *but*
- are computationally expensive, *and*
- require cycle detection, *so*
- good search control heuristics are essential!