Partial Order Planning

Alan Bundy
School of Informatics
(Slides courtesy of Robert Wilensky)

Coping With Interactions

- A big problem with situation-based planning is coping with interacting subgoals.
- E.g., suppose our goal were

  On(A,B) $\wedge$ On(B,C)
- Planning to achieve the first conjunct first is bad, as we will have to undo it before achieving the second and redo it again afterwards.
- No matter how we plan, we must have some mechanism to detect such planning conflicts.
  - Or we will produce buggy or inefficient plans. (STRIPS did the latter.)
- **Moreover, a situation-based planner can only find a solution by blinding searching all possible orderings.**
  - In other words, divide-and-conquer among subgoals doesn't work: Can't just plan for each and then slap the plans together.
  - Called *Sussman Anomaly*.

Alternative: Searching *Plan* Space

- Idea:
  - suggest a *partially specified plan*
  - incrementally refine the plan, until we have a complete plan.
- Result might still be only partially specified plan,
  - But each complete specification of it would be an acceptable solution.
- In effect, we would be searching the space of *plans* rather than of *situations*.
- Most planners today use some variant of this idea.

Kinds of Plan-Space Planners

- Ways in which plans can be partial:
  - Steps are specified, but there is no mention of their order.
    - I.e., plan steps are only partially ordered. (Such plans are called a *non-linear* or *partially-ordered plans*.)
  - Not every variable may be bound.
    - I.e., the plan is not fully instantiated.
- Such planners are sometimes called 'least commitment' planners.
- Plans might be specified 'abstractly', at some coarse level of detail, then planned at successive levels of refinement.
  - Such planners are said to be *hierarchical*.

Partially-Ordered Planning

- Solutions will be
  - Complete, i.e., every precondition is achieved by a step
  - Consistent, i.e., no conflicts in the order of steps or binding of variables
- but may be partial, i.e., it may not be
  - linear
  - fully instantiated
- Every linearized, full-instantiation of complete and consistent solution will also be a complete and consistent solution.

Simplest Example of Non-Linear Plan

- If goal is **On(A,B)** $\wedge$ **On(C,D)**,
  - where everything on table.
  - can plan **{Move(A,B),Move(C,D)}**, i.e., a set of steps without a specified order.
- Linearized plans would then be

  Move(A,B) < Move(C,D)

  and

  Move(C,D) < Move(A,B)

  where **<** indicates order.

Partially-Ordered Planning: Basic Idea

- We'll keep using our STRIPS-style representation for operators.
- We'll introduce a representation for partial plans.
- Start off with some minimal plan for a goal.
- Consider refining the plan.
  - In doing so, consider repercussions of proposed refinement,
  - looking out for possible conflicts.
  - If we find a conflict, try to resolve by constraining the plan.
  - If fail, back off and try another refinement.
- Keep going until the plan is complete.

A Formalization: Plans (or 'Task Networks')

- A set of *steps*
  - A step is just an operator
- A set of *variable binding constraints*
  - of the form ***v=x***, where ***v*** is a variable of some plan step, and ***x*** is either a constant or a variable of another plan step
- A set of *ordering constraints*
  - e.g., $S_1 < S_2$, i.e., step S_1 comes before step S_2.
- A set of *protection intervals*
  - e.g., state **C** must persist prior to step **S** and until after **S** is completed.
  - Alternatively, $S_i \xrightarrow{c} S_j$ is a *causal link*, i.e., step S_i establishes state **C**, satisfying a precondition for step S_j.

A Nice Hack

- We'll Use (pseudo) operators **Start** and **Finish**, precluding the need for goals, initial states, etc.
- **Start** will always have no preconditions, and Add the initial state predications.
- **Finish** will always have the goal conjuncts as its preconditions (and do nothing).

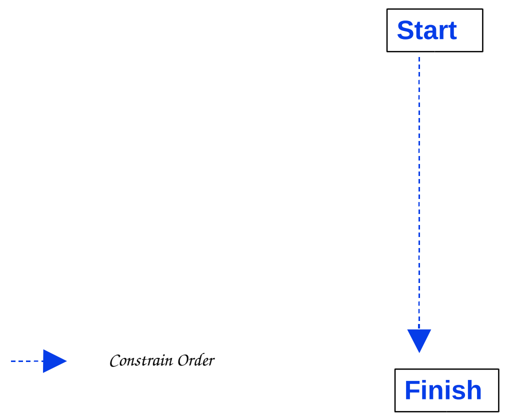
Simple Example

- **Start( )**

  Add: **On(A,Table)** ∧ **On(B,Table)** ∧ **On(C,Table)** ∧ **Clear(A)** ∧ **Clear(B)** ∧ **Clear(C)**
- **Finish( )**

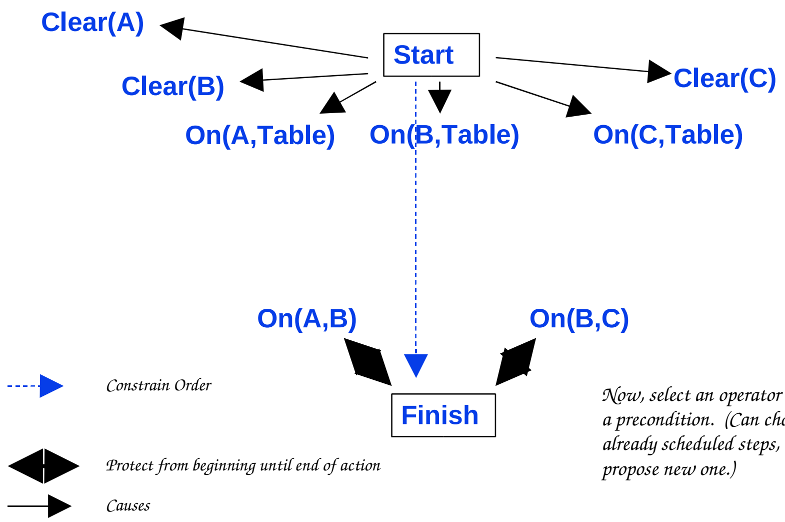
  Pre: **On(A,B)** ∧ **On(B,C)**
- I.e., **Start** just sets up the initial state; **Finish** requires the goal.
- Our initial plan or task network will be:
  - Steps: **{S_1:Start( ), S_2:Finish( )}**
  - Ordering constraints: **{S_1 < S_2}**
  - Protection intervals: **{ }**

Or, Graphically



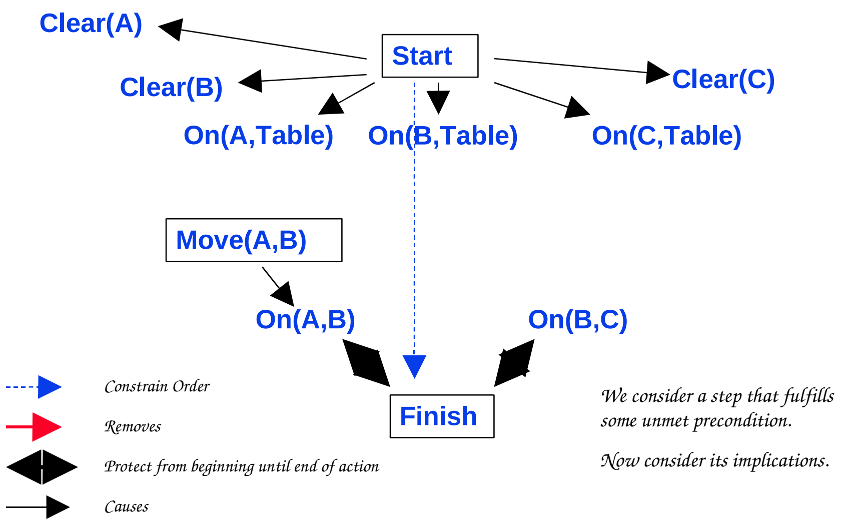
Immediately add Adds of **Start** *and Pre of* **Finish**.

Or, Graphically

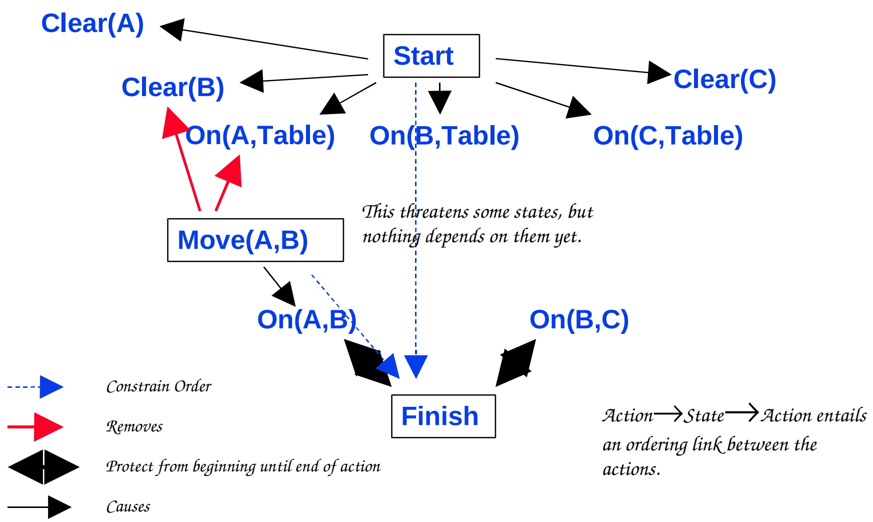


Now, select an operator to fulfill a precondition. (Can choose from already scheduled steps, or propose new one.)

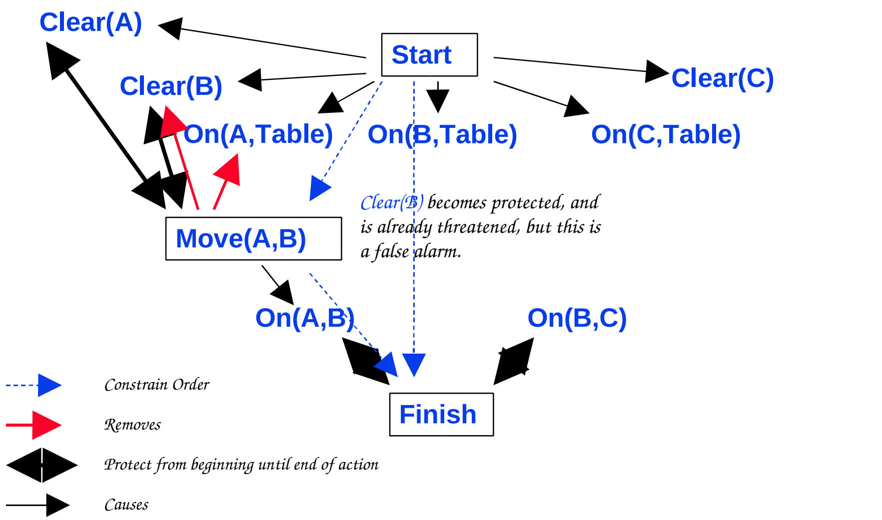
Example (con't): Propose a plan to achieve a precondition



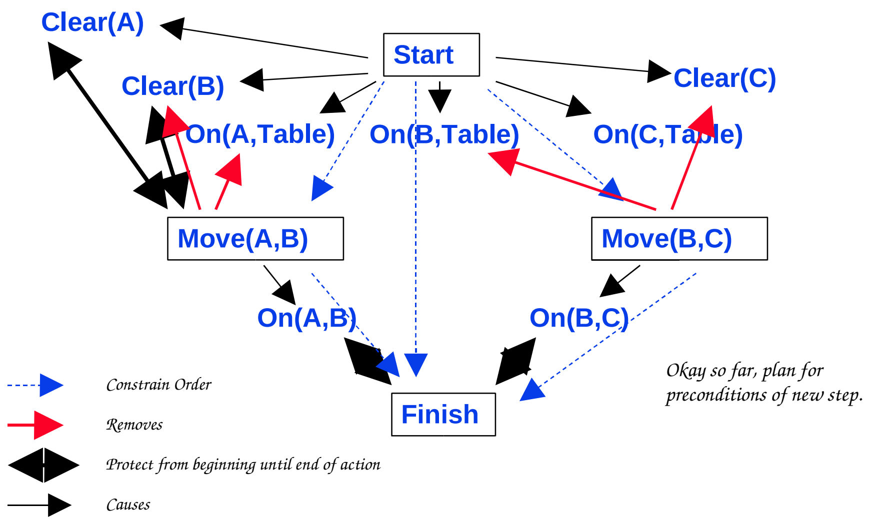
Example (con't): Propose a plan to achieve a precondition



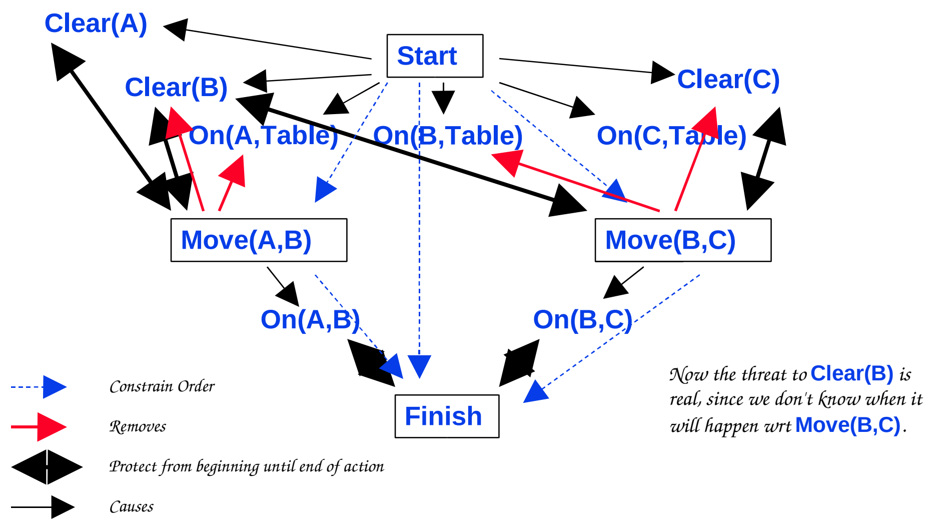
Example (con't): Again, this time using an existing step.



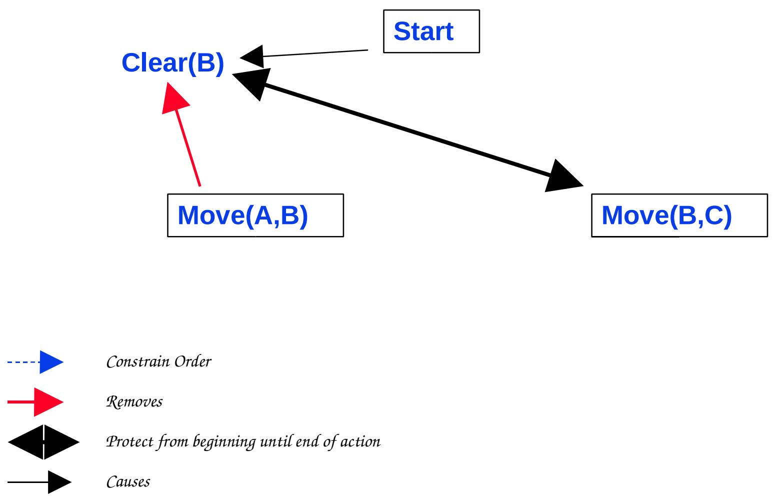
Example (con't): Pick another unmet precondition to achieve



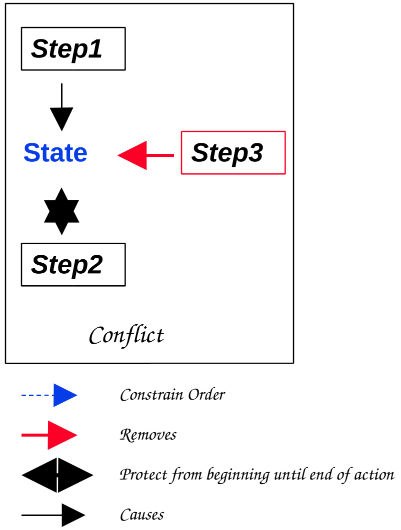
Example (con't): Pick another unmet precondition to achieve



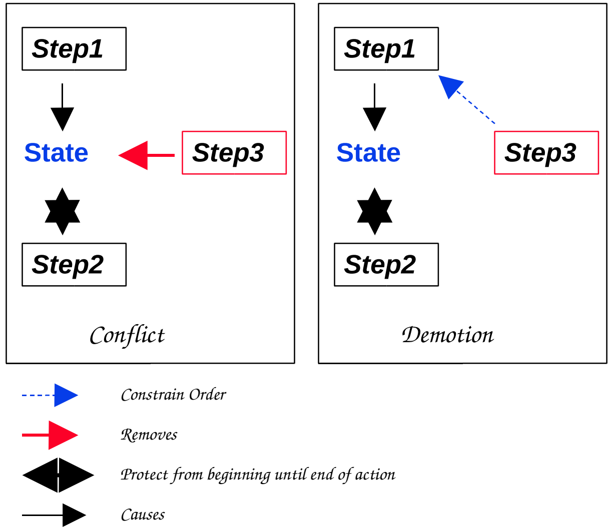
A Protected State is Threatened



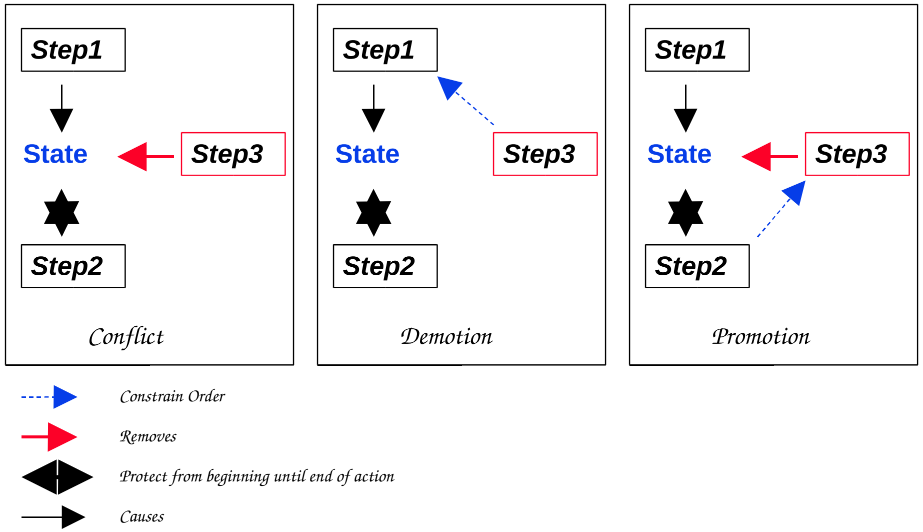
Resolving Protection Violations



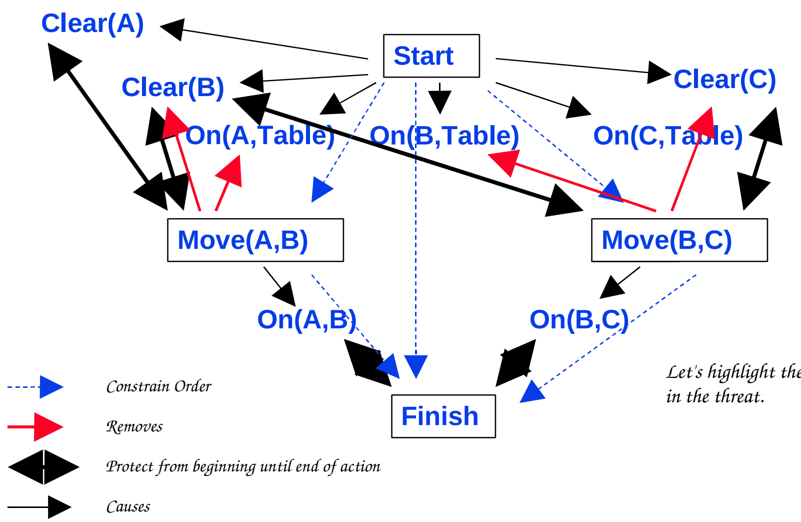
Resolving Protection Violations



Resolving Protection Violations

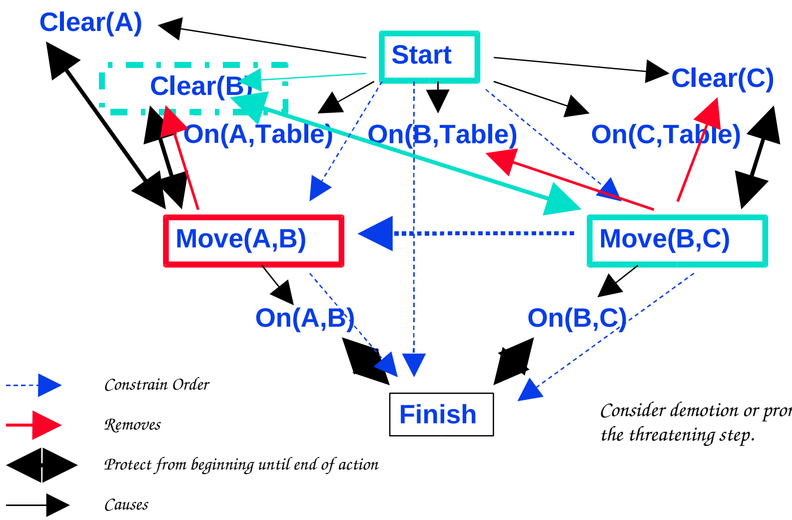


Back to Our Example …



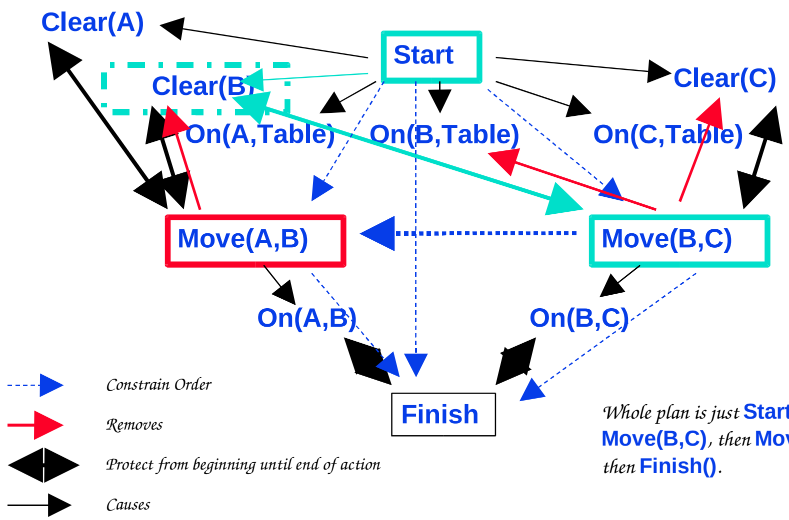
Let's highlight the steps involved in the threat.

Example (con't): Resolve threat by ???



Consider demotion or promotion of the threatening step.

Example (con't): Resolve threat by *promotion*



Whole plan is just **Start()**, *then* **Move(B,C)**, *then* **Move(A,B)**, *then* **Finish()**.

Point

- We dealt with an interaction between goals without any backtracking.
- We won't always be able to avoid backtracking this way, but we generally can avoid much of it.

General Partial Planning Algorithm

- Make initial plan
- Until no more or fail,
  - Select subgoal of step
  - Choose operator (with desired effect)
    - by considering currently scheduled or new plan steps
    - updating plan, noting threatened states, bindings
  - Resolve any threats
    - If real, promote, demote, or fail
- This is a complete planner!

Partial Planning

- Is provably correct and complete.
- Planning is hard.
  - With just simple STRIPS-like operators, planning is not decidable.
  - But it is partially decidable. (I.e., if there is a plan, we can find it.)
  - Finding a findable plan is NP-hard.
    - Actually, just determining whether a state is necessarily true in a partial planner whose action representation is powerful enough to represent conditional actions, dependency of effects on states, or derived side-effects in NP-hard.
- So, if planning is so hard, why is it so easy?