Artificial Intelligence
ICS461
Fall 2010

Nancy E. Reed

nreed@hawaii.edu

Lecture #10A – Classical Planning Outline

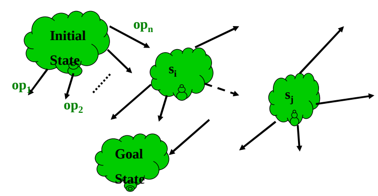
- The Planning problem
- Classical planning
- STRIPS
- Partial Order Planning
- Chapter 10

What is Planning?

- Generate sequences of actions to perform tasks and achieve objectives.
  - States, actions and goals
- Search for solution over abstract space of plans.
- Assists humans in practical applications
  - design and manufacturing
  - military operations
  - games
  - space exploration

Search or Planning Agent?

- State Space Search: (Problem solving agent)



States
Initial State
Goal State
Actions (Operators)

- Planning (Planning agent) – computes several steps of problem solving procedure before executing them. **Plan: sequence of actions.**

Previous Agent Limitations

- Ch 3 model-based agent
  - Uses an atomic representation of states
  - With good domain-specific heuristics
  - can find a goal state not too large
- Ch 7 hybrid propositional logic agent
  - may be swamped with large search spaces having many actions and/or states

Planning: Example

- Goal: *Get a bunch of bananas, a cordless drill, and a quart of milk*
  - Initial State: **At Home $\wedge$ No milk $\wedge$ No cordless drill $\wedge$ No bananas**
  - Goal State: **At Home $\wedge$ milk $\wedge$ cordless drill $\wedge$ bananas**

Problem: Many actions and many states to consider; sequence of actions could become too long

$PlanResult(p, s)$ is the situation resulting from executing p in s

$PlanResult([\,], s) = s$

$PlanResult([a|p], s) = PlanResult(p, Result(a, s))$

Initial state $At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \ldots$

Actions as Successor State axioms

$Have(Milk, Result(a, s)) \Leftrightarrow$

$[(a = Buy(Milk) \wedge At(Supermarket, s)) \vee (Have(Milk, s) \wedge a \neq \ldots)]$

Query

$s = PlanResult(p, S_0) \wedge At(Home, s) \wedge Have(Milk, s) \wedge \ldots$

Solution

$p = [Go(Supermarket), Buy(Milk), Buy(Bananas), Go(HWS), \ldots]$

Principal difficulty: unconstrained branching, hard to apply heuristics

Tidily arranged actions descriptions, restricted language

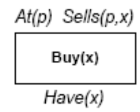
ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

[Note: this abstracts away many important details!]

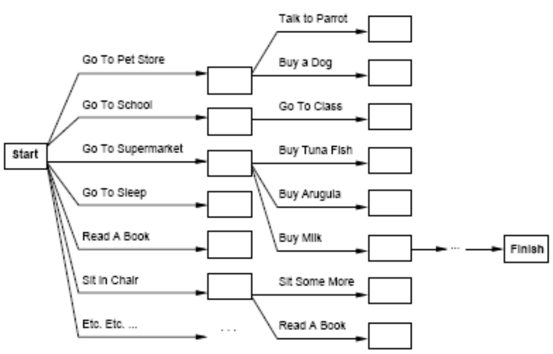
Restricted language $\Rightarrow$ efficient algorithm

Precondition: conjunction of positive literals

Effect: conjunction of literals



Standard Search Algorithms seem to Fail Miserably



Forward vs. Backward Search

- Forward search may not succeed.
  - Starting at initial state.
    - Forces agent's hand.
    - *Example: Since agent is at home, and the things he needs are not at home, he must go somewhere. But where?*
- Backward search may be better.
  - *One way to have milk is to buy it.*
  - *Milk can be purchased at the grocery store. Therefore, …..*

Planning

- Determine possible next states by applying evaluation function.
  - Only select states closer to goal.
- **Problem: Too many states!**
  - **Goal and evaluation function are black boxes, therefore, cannot really <u>eliminate a number of actions from consideration</u>.**
- Note: Evaluation function is being applied after the fact
- After the fact goal evaluation isn't enough

Search vs. Planning

Planning systems do the following:

1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

	Search	Planning
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

Planning: Classical Approach

- Considers environments that are:
  - Fully observable
  - Deterministic
  - Finite
  - Static
    - Only change is via the agent's actions
  - And are discrete in time, action, objects, and effects.

Planning Languages

- What is a good language?
  - **Expressive** enough to describe a wide variety of problems.
  - **Restrictive** enough to allow efficient algorithms to operate on it.
  - Planning algorithm should be able to take advantage of the **logical structure** of the problem.
- STRIPS and ADL

General Language Features

Representation of states

- Decompose the world in logical conditions and represent a state as a *conjunction of positive literals.*
  - Propositional literals: *Poor* $\wedge$ *Unknown*
  - FO-literals (grounded and function-free): *At(Plane1, Melbourne)* $\wedge$ *At(Plane2, Sydney)*
- Closed world assumption

Representation of goals

- Partially specified state and represented as a *conjunction of positive ground literals*
- A goal is *satisfied* if the state contains all literals in goal.

General Language Features

Representations of actions

- Action = PRECOND + EFFECT

  Action(Fly(p,from, to),

  PRECOND: At(p,from) $\wedge$ *Plane(p)* $\wedge$ *Airport(from)* $\wedge$ *Airport(to)*

  EFFECT: $\neg$*AT(p,from)* $\wedge$ *At(p,to))*

  = action schema (p, from, to need to be instantiated)
  - Action name and parameter list
  - Precondition (conj. of function-free literals)
  - Effect (conj of function-free literals and P is True and not P is false)
- Add-list vs. delete-list in Effect

How Do Actions Affect States?

- An action is applicable in any state that satisfies the precondition.
- For FO action schema applicability involves a substitution θ for the variables in the PRECOND.
  - At(P1,JFK) $\wedge$ At(P2,SFO) $\wedge$ Plane(P1) $\wedge$ Plane(P2) $\wedge$ Airport(JFK) $\wedge$ Airport(SFO)
  - Satisfies : At(p,from) $\wedge$ Plane(p) $\wedge$ Airport(from) $\wedge$ Airport(to)
  - With θ ={p/P1,from/JFK,to/SFO}
  - Thus the action is applicable.

STRIPS

- (**ST**anford **R**esearch **I**nstitute **P**roblem **S**olver) language
  - Relies on representing states, goals and actions.
  - Uses Closed World Assumption in state representation.

STRIPS

- **States:** Conjunction of grounded function free predicates
  - States are always represented with positive literals (no negations).

  Initial State:

$$At(Home) \wedge Want(Milk) \wedge Want(Bananas) \wedge Want(Drill)$$

- **Goals:** A partially specified State.

$$At(Home) \wedge Have(Milk) \wedge Have(Bananas) \wedge Have(Drill)$$

STRIPS Actions: Represented by

- Precondition: conjunction of function free positive literals that must be true before the action is applied.
- Add list: A set of function free literals that will be added to the current state.
- Delete List: A set of function free literals that will be deleted from the current state.
- Effect: Conjunction of function free literals representing how the state changed when the action is applied.
  - Negations in Effect list represent items to be deleted.

STRIPS Operators

- Restricted Language

  Action: *Buy(x)*
  Precondition: *At(p)* $\wedge$ *Sells(p,x)* $\wedge$ *Want(x)*
  Add List: *Have(x)*
  Delete List: *Want(x)*
  Effect: $\neg$Want(x) $\wedge$*Have(x)*

- abstracts away many important details $\Rightarrow$ search algorithm to generate efficient plan.

STRIPS

STRIPS action consists of three parts

1. *PC: preconditions*
2. *D: delete list (negative literals)*
3. *A: add list (positive literals)*
4. *Ef: Effect*
   - *After-action* state description generated by deleting all literals in *D* from the *before-action* state description, and then add the literals in *A*.
   - **Strips Assumption:** All literals not mentioned in *D* carry over from the *before-action* to the *after-action* state (Effect). -- Important: ***Frame Problem.***

Executing Actions

The result of executing action a in state s is the state s'

- s' is same as s except
  - Any positive literal *P* in the effect of *a* is added to *s'*
  - Any negative literal ¬*P* is removed from *s'*

  At(P1,SFO) ∧ At(P2,SFO) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO)

- STRIPS assumption: (avoids representational frame problem)

  every literal NOT in the effect remains unchanged

Expressiveness and Extensions

- STRIPS is simplified
  - Important limit: function-free literals
  - Allows for propositional representation
- Function symbols lead to infinitely many states and actions
- Recent extension:Action Description language (ADL)

  Action(Fly(p:Plane, from: Airport, to: Airport),
  PRECOND: At(p,from) ∧ (from ≠ to)
  EFFECT: ¬At(p,from) ∧ At(p,to))

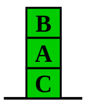
Standardization : *Planning domain definition language (PDDL)*

STRIPS applied to the Blocks World

- State Description:

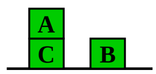
$On(B,A) \wedge On(A,C) \wedge On(C,Fl) \wedge Clear(B) \wedge Clear(Fl)$

- STRIPS operator:



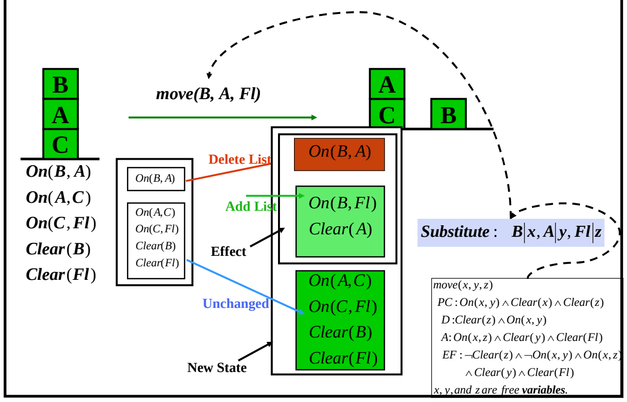
$move(x,y,z)$
$PC: On(x,y) \wedge Clear(x) \wedge Clear(z)$
$D: Clear(z) \wedge On(x,y)$
$A: On(x,z) \wedge Clear(y) \wedge Clear(fl)$
$Ef: \neg Clear(z) \wedge \neg On(x,y) \wedge On(x,z) \wedge Clear(y) \wedge Clear(fl)$
$x, y,$ and z are free **variables**.

Apply it to given state: $move(B,A,Fl)$

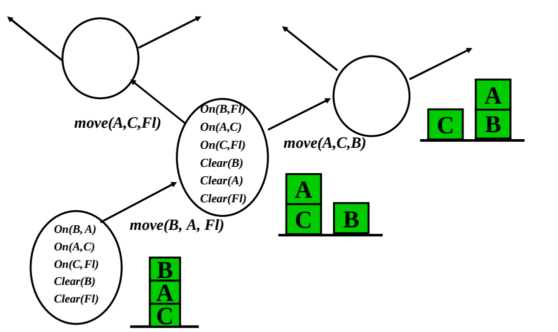


Then, *move(A,C,Fl)* or *move(A,C,B)*, etc.

STRIPS Applied to the Blocks World



STRIPS Algorithm: Forward Search

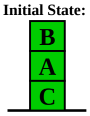


STRIPS Algorithm

- Forward search (progression planning) with a divide-and conquer heuristic.
  - Solve conjunctive goal literals, one at a time.
  - Based on the *General Problem Solver (GPS)* developed by Newell, Shaw, and Simon in 1959
    - Technique to solve goal literal called *Means-end analysis*.

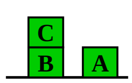
Apply STRIPS Algorithm

Initial State:



$On(B,A) \wedge On(A,C) \wedge On(C,Fl) \wedge Clear(B) \wedge Clear(Fl)$

Goal State:



$On(A,Fl) \wedge On(B,Fl) \wedge On(C,B)$

No initial state description meets goal constructs.

Divide and conquer: choose $On(A,Fl)$ as initial goal.

***move(A,x,Fl)* has *On(A,Fl)* in its add list.**

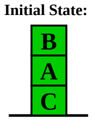
Call STRIPS recursively to achieve its preconditions: $Clear(A) \wedge Clear(Fl) \wedge On(A,x)$

Make substitution C/x, which makes all but *Clear(A)* satisfied by initial state

$move(x,y,z)$
$PC: On(x,y) \wedge Clear(x) \wedge Clear(z)$
$D: Clear(z) \wedge On(x,y)$
$A: On(x,z) \wedge Clear(y) \wedge Clear(Fl)$
$EF: \neg Clear(z) \wedge \neg On(x,y) \wedge On(x,z)$
$\wedge Clear(y) \wedge Clear(Fl)$
$x, y,$ and z are free **variables**.

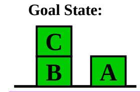
Apply STRIPS Algorithm

Initial State:



$On(B,A) \wedge On(A,C) \wedge On(C,Fl) \wedge Clear(B) \wedge Clear(Fl)$

Goal State:



$On(A,Fl) \wedge On(B,Fl) \wedge On(C,B)$

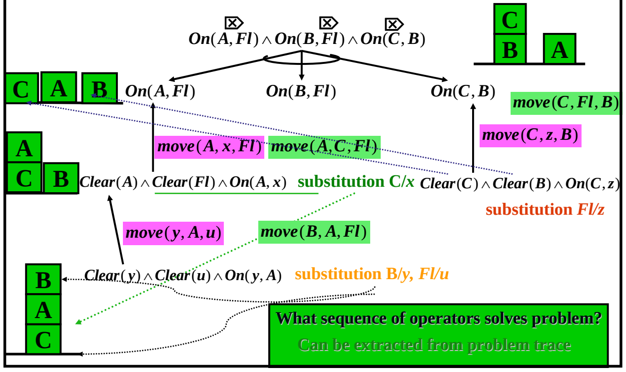
Call STRIPS again (recursively) to solve *Clear(A)*

Use operator *move(y,A,u)* $On(A,Fl)$

Precondition: $Clear(y) \wedge Clear(u) \wedge On(y,A)$

Continue this process and solve the problem.

Apply STRIPS Algorithm



STRIPS Algorithm

STRIPS (γ) -- γ is conjunctive goal formula

Uses global data structure, ***S***, consisting of a set of ground literals – **<u>initial state description</u>**.

1. **Repeat** (iterative main loop continues till state description = γ) Termination in step 9 produces a substitution σ, such that some conjuncts (possibly none) of γσ appear in *S*. Several substitutions can be tried in performing the test, so this is *<u>possible backtracking point</u>*.
2. ***g*** **← an element of γσ , such that *S*** $\vert\neq$ ***g***. In **<u>means-end analysis</u>** terms *g* is regarded as "difference" that must be "reduced" to achieve the goal. (*<u>Backtracking point</u>*).
3. ***f*** **← STRIPS rule whose add list contains the literal, λ, that unifies with *g* using unifier *s***. *f* is an operator that is relevant to reducing the difference. More than one choice of *f* : *<u>backtracking point</u>*.
4. ***f' ← fs*. This is an instance of *f* after substitution *s***. *f'* can contain variables (is not a ground instance).
5. ***p*** **← precondition formula for *f'*** (instantiated with the substitution *s*).
6. STRIPS(*p*). **<u>Recursive call with new sub-goal.</u>**
7. *f''* ← a ground instance of *f'* applicable in ***S***
8. ***S*** ← result of applying *f''* to ***S***.
9. **until *S*** $\vert=$ γ. (**<u>This test is always against the entire goal γ</u>**).

STRIPS Plans can Have Problems

- Example: Sussman Anomaly

Initial State: $on(C, A) \wedge on(B, Fl) \wedge On(A, Fl)$ **Goal condition:** $on(A, B) \wedge on(B, C)$

Question: Can STRIPS solve this problem?

If not why not?

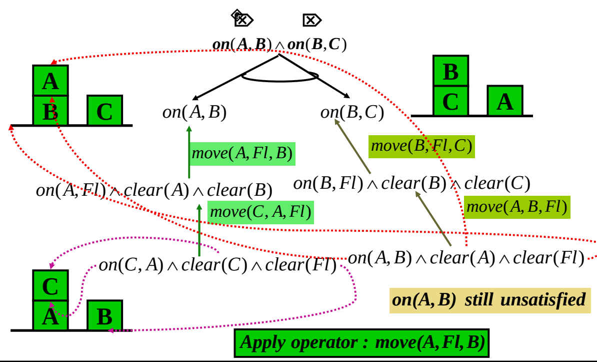
Divide and conquer approach:

First solve $on(A, B)$ then $on(B, C)$

What happens?

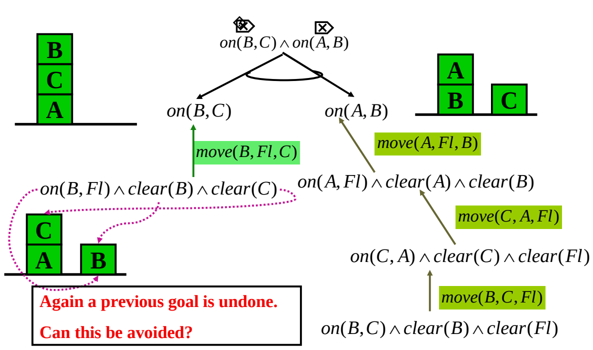
Problem of focusing narrowly on one conjunct at a time

Sussman Anomaly



Sussman Anomaly:

Solve by changing goal order



Sussman Anomaly (notes)

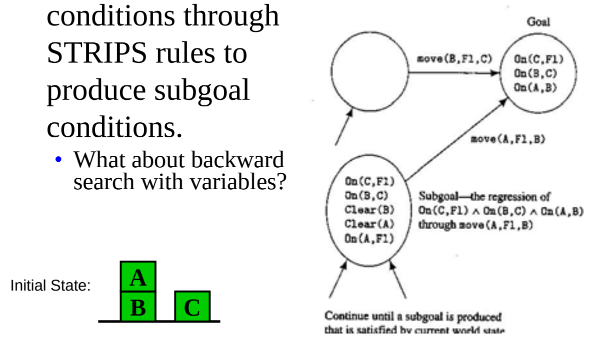
- STRIPS **can generate a plan** for Sussman anomaly situations but has to undo a goal condition to satisfy the other goal condition.
- There is no way STRIPS can avoid this, irrespective of which goal condition it starts of with first.

Backward Search

- Search backwards from goal states.
- Also known as **regression planning**
- Considers only **relevant** actions.
  - Asks: “What are the states from which applying a given action leads to the goal?”
  - What does this consideration imply about the search space?
- Any action that does not undo a desired literal is **consistent**
  - All actions must be consistent.

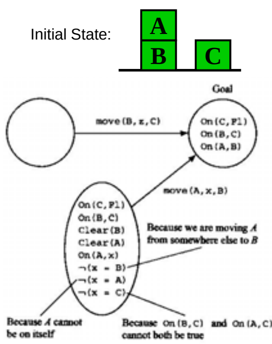
Regression Planning

- ***Regress*** goal conditions through STRIPS rules to produce subgoal conditions.
  - What about backward search with variables?

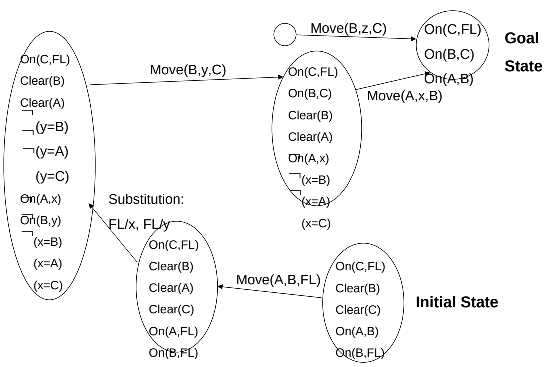


Regression Planning

- Why move A from floor to B?
- So why make commitment?
  - Principle: **Least Commitment Planning**
  - Regress goal through partially instantiated operators
  - Makes backward search complicated, but avoids problems, such as Sussman Anomaly



Regression Planning



Total Order Searches

Forward and backward searches discussed are **Total Order searches**

- Strictly linear sequence of actions from start to goal states.
- Does not permit independent sub-goal problem solving
- Is this realistic for real world problems?
- How can we deal with these issues?

Planning: Flexible Order

- Need representations for states, goals, and actions.
- Planner can add actions to the plan in any order as needed.
  - Do not have to start at initial state and incrementally (sequentially) progress.

Partial Order Plans

Partially ordered collection of steps with
- *Start* step has the initial state description as its effect
- *Finish* step has the goal description as its precondition
- causal links from outcome of one step to precondition of another
- temporal ordering between pairs of steps

Open condition = precondition of a step not yet causally linked

A plan is complete iff every precondition is achieved

A precondition is achieved iff it is the effect of an earlier step
and no possibly intervening step undoes it

Partial Order Plans (2)

Operators on partial plans:
- add a link from an existing action to an open condition
- add a step to fulfill an open condition
- order one step wrt 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

Planning: Problem Decomposition

- Many parts of the world are independent of others.
- Use ***divide and conquer***.
  - Create sub-problems and solve them one by one.
  - Many planners assume subgoals can be solved independently.
    - Will this approach always work?

Partial Order Planning

- Allows planning for multiple independent sub-goals.
- The sub-plans are then combined to form a complete plan.
- Allows one to solve *obvious* and *important* decisions first
- "Any planner that can place two actions into a plan without specifying which comes first is a partial order planner."

Partial Order Planning Components

Each POP has four components

1. The plan **action** steps.
2. The **ordering constraints** determine in what order steps are to be executed.
   - Ordering constraints that create cycles are **not** added to plans.
   - Fewer ordering constraints simplify plan construction.

POP Components (cont)

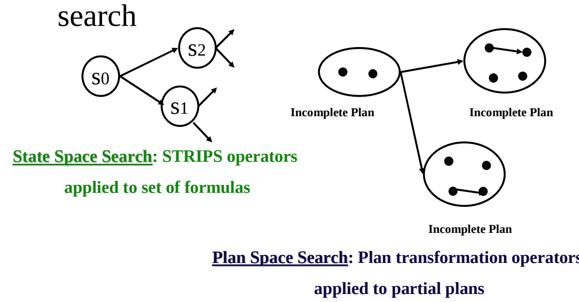
3. A set of **causal links** indicates what assertions must be true between two actions.
   - "A achieves p for B." $A \xrightarrow{p} B$
   - p must be true from the time A is completed and B starts.
4. The set of **open preconditions** contains all preconditions that have not yet been achieved by an action in the plan.
   - Planners attempt to empty the set while not introducing contradictions.

Consistency

- A **consistent** *plan* contains no cycles in the ordering constraints and no conflicts with the causal links.
- A **solution** is a consistent plan with an empty precondition set.

Plan Spaces

- Introduce the notion of plan space search

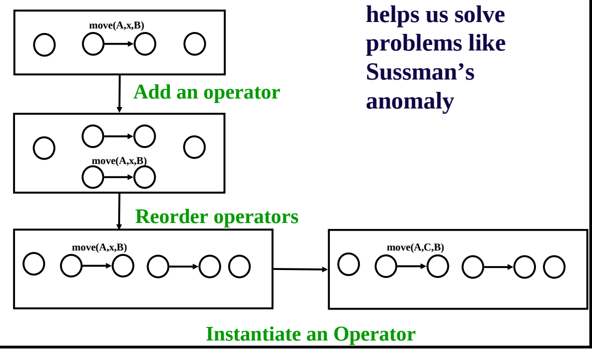


Plan Transformations

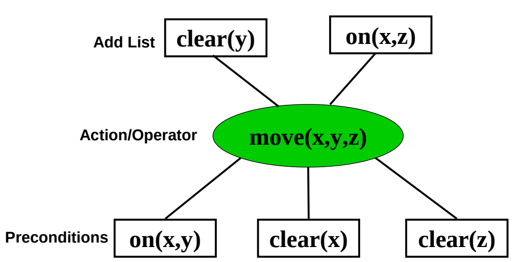
- Potential plan transformations
  - Adding steps to the plan.
  - Reordering existing plan steps.
  - Changing a POP into a fully-ordered plan.
  - Changing the plan schema (with instantiated variables) into some instance of that schema.
- Plan Transformations are considered actions in the search, NOT actions on the world.

Plan Transformation

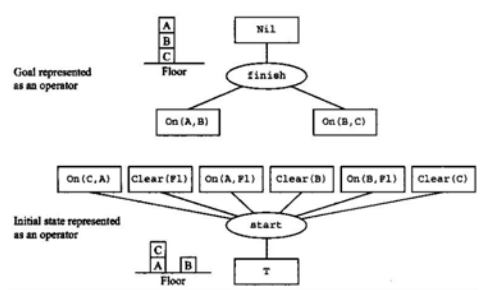
Let's see if this helps us solve problems like Sussman's anomaly



STRIPS Rule: Graphical Representation



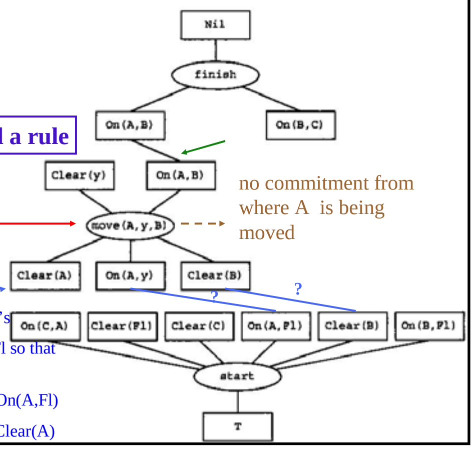
Partial Order Planning (POP)



Start with the start and finish rules

Plan incomplete: apply plan transformation operators

Step 1: Add a rule



no commitment from where A is being moved

What next?

(i) link the Clear(B)'s

(ii) instantiate y to Fl so that on(A,y) can be connected to On(A,Fl)

(iii) Try and satisfy Clear(A)

?

Step 2: Add a rule

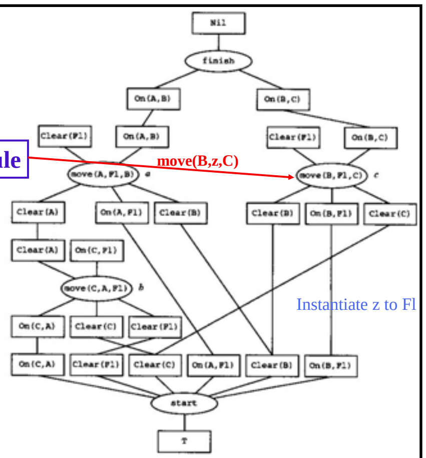
Next: solve this goal

move(u,A,v)

All instantiations performed, so preconditions of all rules Satisfied

Implied ordering: *b* before *a* (*b* < *a*)

Step 3: Add a rule



move(B,z,C)

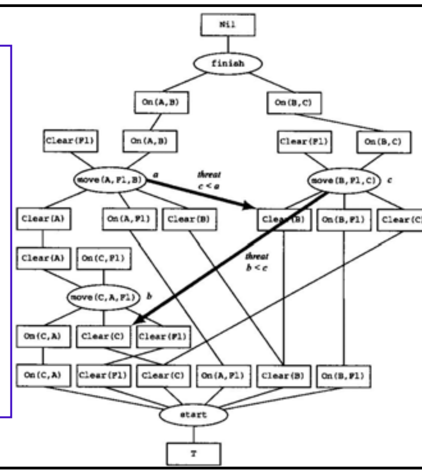
Instantiate z to Fl

We know: *b* before *a* (*b* < *a*)

What about c?

Currently no commitment – plan is partially ordered

Step 4: Look for threat arcs



Two threat arcs in this plan

Order operators to avoid clobbering

c <a and b < c

Also, from graph it is clear that b < a

Therefore, we can derive

The total order:

b < c < a

Advantages/Disadvantages

- Advantage:
  - POP is able to decompose problems into subproblems.
- Disadvantage:
  - POP does not directly represent states
  - It is harder to determine how close the POP is to achieving the goal.

60

Planning, part B

- Continuation of classical planning in next set of slides