TDT4136 Introduction to Artificial Intelligence

Lecture 11 - PLanning

Chapter 11 in textbook

Pinar Özturk

Norwegian University of Science and Technology

2021

Outline

- Planning representation languages
- Classical Planning in state-space
  - forward
  - backward
- Planning in plan space
  - Partially ordered plans
- Heuristic for classical planning
- Planning under Uncertainty
  - Planning of Sensorless agents
  - Contingenct planning in partially observable and/or nondeterministic environments

What is planning?

Human Planning and Acting

- Acting without (explicit) planning
  - when purpose is immediate
  - when performing well-trained behaviour
  - when course of action can be freely adopted

What is planning?

Human Planning and Acting

- Acting without (explicit) planning
  - when purpose is immediate
  - when performing well-trained behaviour
  - when course of action can be freely adopted
- Acting after planning
  - when addressing a new situation
  - when tasks are complex
  - when the environment imposes high risk/cost
  - when collaborating with others

People plan only when strictly necessary.

An early example

- Example: NASA's Deep Space 1
  - Launched in 1998 to test technologies and perform flybys of asteroid Braille and Comet Borrelly
  - First spacecraft to be controlled by an AI system without human intervention
  - Remote Agent (remote intelligent self-repair software) system used to plan on-board activities and correctly diagnose and respond to simulated faults
  - Planning system was later used on the ground-based Mars Exploration Rovers

http://ti.arc.nasa.gov/tech/asr/planning-and-scheduling/remote-agent/

How to represent states, actions, goal

- We can consider planning as problem solving on the state space, i.e, through search (e.g., A*)
  - Then we represent nodes as states, edges as actions, and a plan as a path, i.e., a sequence of actions
  - We need a domain specific heuristic for each domain
- We can do planning using logic
  - Then we need to encode a lot of axioms to capture the effects of the actions.

Issues related to logic representation in planning

- The time is an important aspect in Planning. What is true when?
- An action changes some aspects of the world (hence fluents), but not all aspects.
  - Definition: We refer to aspects of world that changes from one time step to another as **fluents**.
  - Example fluent: Location_{xy}^{t} = location, i.e, coordinates of the square the agent is in at time t is a fluent.
  - The agent needs to keep track of fluents. But it should also know what remains unchanged

Representation related issues

Let us revisit Chapter 7, Wumpus example.

- It is important to note that a percept is about only one time step.
  - For example, proposition **Stench** now may be contradictory if $\neg$ **Stench** is already in the KB - if we don't represent time.
  - There is Stench in time step 3, i.e, $\textbf{Stench}^3$ will not contradict with $\neg$ $\textbf{Stench}^2$

Representation related issues

Let us revisit Chapter 7, Wumpus example.

- It is important to note that a percept is about only one time step.
  - For example, proposition **Stench** now may be contradictory if $\neg$ **Stench** is already in the KB - if we don't represent time.
  - There is Stench in time step 3, i.e, **Stench**^{3} will not contradict with $\neg$ **Stench**^{2}
- The *effect* of an action makes change in the **fluents**.
  - Example **effect axiom**: The agent is at location/square [1,1] and facing east at time=0 and goes forward. The result is that the agent is in square [2,1] and not any longer in [1,1] at time=1:

    - $\text{Location}_{11}^0 \wedge \text{FacingEast}^0 \wedge \text{Forward}^0 \implies \text{Location}_{21}^1 \wedge \neg \text{Location}_{11}^1$

Representation related issues

Let us revisit Chapter 7, Wumpus example.

- It is important to note that a percept is about only one time step.
  - For example, proposition **Stench** now may be contradictory if $\neg$ **Stench** is already in the KB - if we don't represent time.
  - There is Stench in time step 3, i.e, **Stench**^{3} will not contradict with $\neg$ **Stench**^{2}
- The *effect* of an action makes change in the **fluents**.
  - Example **effect axiom**: The agent is at location/square [1,1] and facing east at time=0 and goes forward. The result is that the agent is in square [2,1] and not any longer in [1,1] at time=1:

    - $\text{Location}_{11}^0 \wedge \text{FacingEast}^0 \wedge \text{Forward}^0 \implies \text{Location}_{21}^1 \wedge \neg \text{Location}_{11}^1$
- If we use effect axioms, we'll need explicit assertion of such sentences about all time steps, all squares, actions, etc
- In addition, effect axioms don't represent/express what remains unchanged by an action - **representational frame problem**

The Frame Problem.

- So, the agent needs to know what remains unchanged in the environment when a particular action is executed.
- A solution: The unchanged aspects of world can be represented in form of "Frame axioms[1]" expressing all propositions that remain unchanged. E.g.,
  - $\text{Forward}^t \implies (\text{HaveArrow}^t \Leftrightarrow \text{HaveArrow}^{t+1})$
  - $\text{Forward}^t \implies (\text{WumpusAlive}^t \Leftrightarrow \text{WumpusAlive}^{t+1})$
  - ...
- Too many such axioms, hence inefficient

[1] Remember that axioms are the sentences that are assumed true, not needing to be proven

Preconditions for Actions

- Representation of actions also need explicit statement of when an action is executable
- I.e., the conditions where the action can be applied, as a **precondition axiom**
- Example in Propositional logic: In order to shoot an arrow, the agent must have an arrow: $\text{Shoot}^t \Leftrightarrow \text{HaveArrow}^t$

Preconditions for Actions

- Representation of actions also need explicit statement of when an action is executable
- I.e., the conditions where the action can be applied, as a **precondition axiom**
- Example in Propositional logic: In order to shoot an arrow, the agent must have an arrow: $\text{Shoot}^t \Leftrightarrow \text{HaveArrow}^t$
- Another issue: the interference between actions
  - For ex. Let us assume that Shoot and Forward may not be done at the same time - How to represent this in the system?
  - Explicitly state every action pair not allowed to "execute" at the same time in form of **action exclusion axioms**
  - Example action execution axiom: $\neg \text{Shoot}^t \vee \neg \text{Forward}^t$
- We will see different types of action (and action-related knowledge) representations in the research field of Planning - soon

Representation language PDDL

- PDDL: Planning Domain Definition Language
- Uses a restricted form of predicate logic
- Representation of States:
- Each state is represented as a conjunction of fluents[2] that are ground, functionless, and non-negative atoms.
- Employs Database semantics, e.g., *closed-world assumption*, i.e. any fluent not mentioned in a state is false
  - The following are not allowed in a state:
    - University(x); x is a university. Not state because of variable.
    - ¬ BestUniversity(NTNU, Norway); not, because of negation.
    - InTrondheim(BestUniversity(Norway)); not, because of function.

[2]Remember fluents are literals representing aspects that change, and literal is an atomic sentence, i.e., a proposition in propositional logic and a predicate in FoL

Representation in PDDL

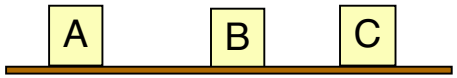
- An initial state is a conjunction of ground atoms - i.e no variable
- A goal is a conjunction of literals (positive/negative) that may contain variables - similar to preconditions.
- Examples

  In(p, Trondheim) $\wedge$ Politician(p), a/any politician is in Trh

  At(x, Trondheim) $\wedge \neg$ At(y, Trondheim)

Representation of States in PDDL and Closed Word Assumption

Representation of states rely on **Closed World Assumption**: any fluent not mentioned in a state is false



Initial state:
On(A,Table),
On (B, Table),
On(C, Table)

These are implicit in the representation, don't need to be explicitly represented:

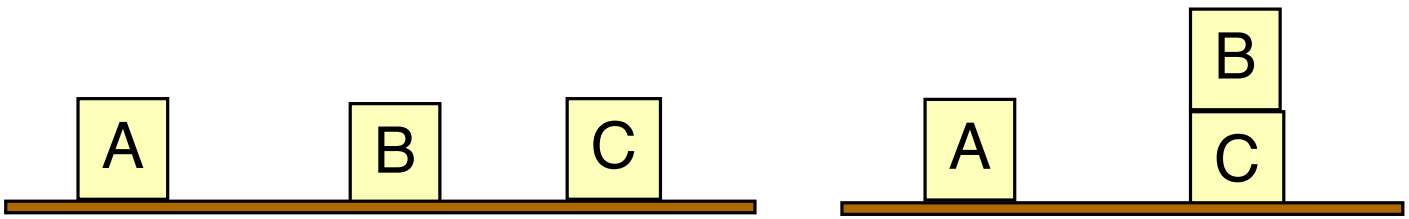
¬On(A,A)
¬On(A,B)
¬On(A,C)
¬On(B,A)
¬On(B,B)
¬On(B,C)
¬On(C,A)
¬On(C,B)
¬On(C,C)
¬On(A,A)
¬On(A,B)
¬On(A,C)
¬On(Table,A)
¬On(Table,B)
¬On(Table,C)
¬On(Table,Table)

Representation of actions in PDDL

- Actions are described at an abstract level as **action schemas**
- An action schema has a precondition and an effect component - that may have variables, and positive/negative literals.
- Times and states are implicit in the action schemas: preconditions refer to time t and effect to time $t + 1$
- The set of action schemas define the domain of the considered planning problem

Representation of actions in PDDL

- Actions are described at an abstract level as **action schemas**
- An action schema has a precondition and an effect component - that may have variables, and positive/negative literals.
- Times and states are implicit in the action schemas: preconditions refer to time t and effect to time $t + 1$
- The set of action schemas define the domain of the considered planning problem
- Representation of actions in the *Block World*; objects are the blocks (i.e, A, B, C) and the Table.



$Action(Move(b, x, y)$
PRECOND : $On(b, x) \wedge Clear(b) \wedge Clear(y)$
EFFECT : $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$

- Moves b from x to y

The Representational Frame Problem

- What is**not** changing in the world as a result of an action?
- It is implicitly assumed that any symbol not mentioned in **effect** remains unchanged.

The Representational Frame Problem

- What is**not** changing in the world as a result of an action?
- It is implicitly assumed that any symbol not mentioned in **effect** remains unchanged.
- How to compute the result of an action *a* taken in state *s* :
  - Effect of an action consisting of the positive and negative effects:

    Effect: On(r,y) $\wedge$ Clear(x) $\wedge \neg$ On(r,x) $\wedge \neg$ Clear(y)

  - The positives (call "ADD") are the ones that will be true in the resulting state while the negative (call "DEL") ones will be removed from state *s*:
  - State of $s+1$ resulted by execution of action *a*:
    RESULT(a, s) = (s - DEL(a)) $\cup$ ADD(a).

Planning Algorithms

Different planning algorithms have different search spaces

- State-space planning

  - Each node represents a state of the world
  - A plan is a path through the space

- Plan-space planning

  - Starts with partial incorrect plan, then apply changes to correct it
  - Each node is a set of partially-instantiated operators, plus some constraints
  - Impose more and more constraints, until we get a plan

State-space planners

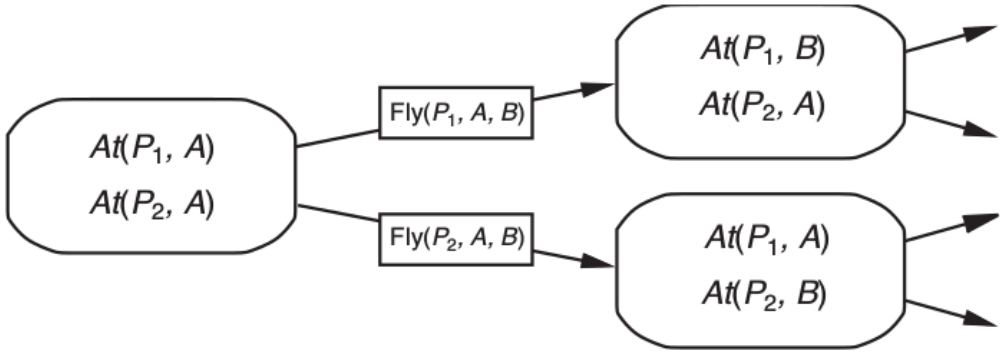
Now we are back to thinking planning as Problem Solving.

Two types of State-space planners:

Progression planners reason from the start state, trying to find the actions that can be applied (match preconditions)

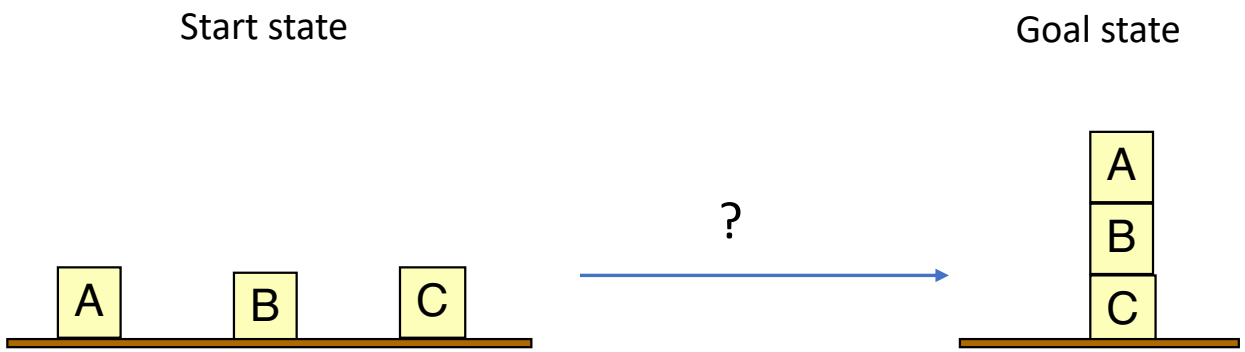
Regression planners reason from the goal state, trying to find the actions that will lead to the start state (match effects)

Progression (forward) planning



1. Ground the action schemas by substituting any variable with constants
   - instantiate the schemas by unifying the variables with the objects/constants in the domain -all possible ways
2. Choose a ground action that is applicable
   - an action of which preconditions satisfied by the current state
3. Determine the new content of the knowledge base, based on the action description
4. Repeat until goal state is reached

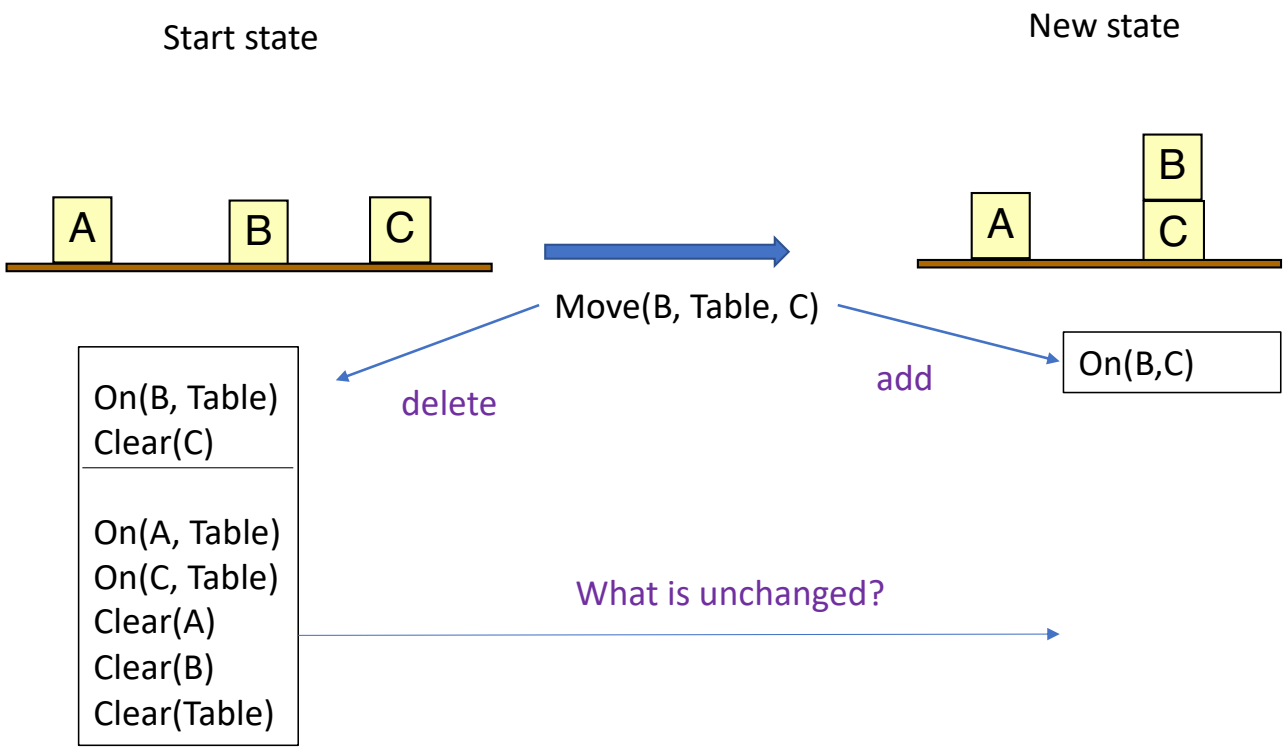
Example: Forward Chaining in Block World



What is your plan?

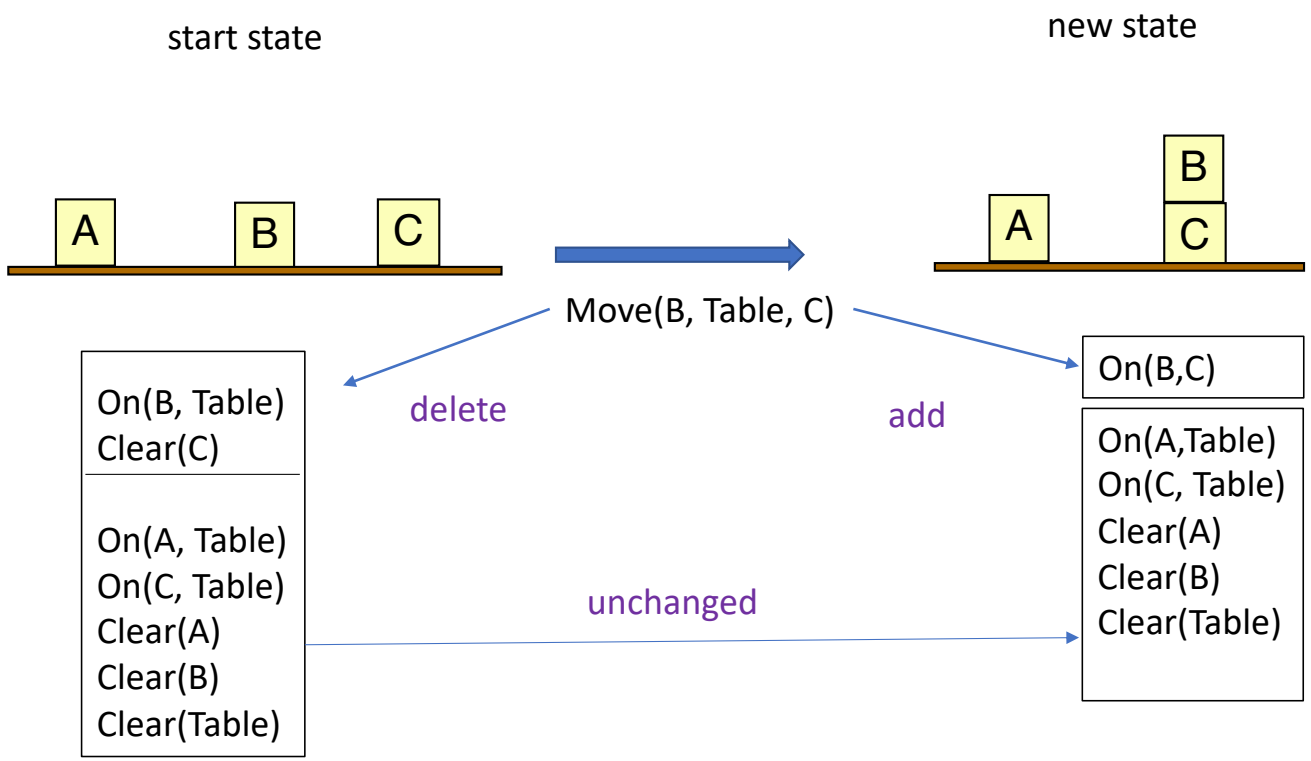
Example: Forward Chaining in Block World -1

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)



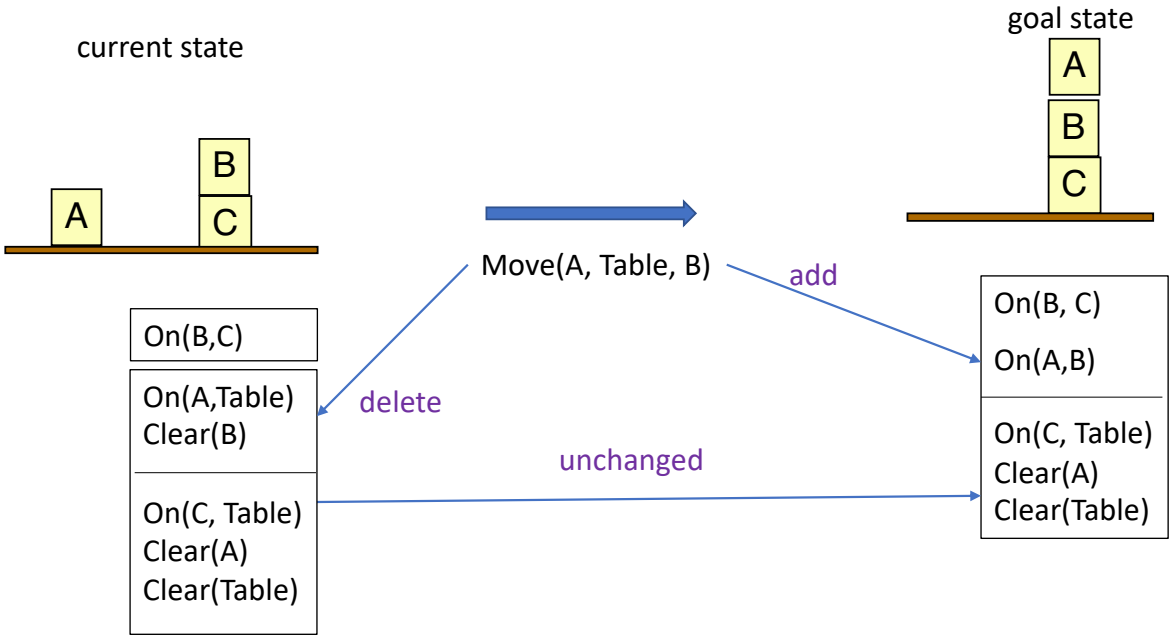
Example: Forward Chaining in Block World-2

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)

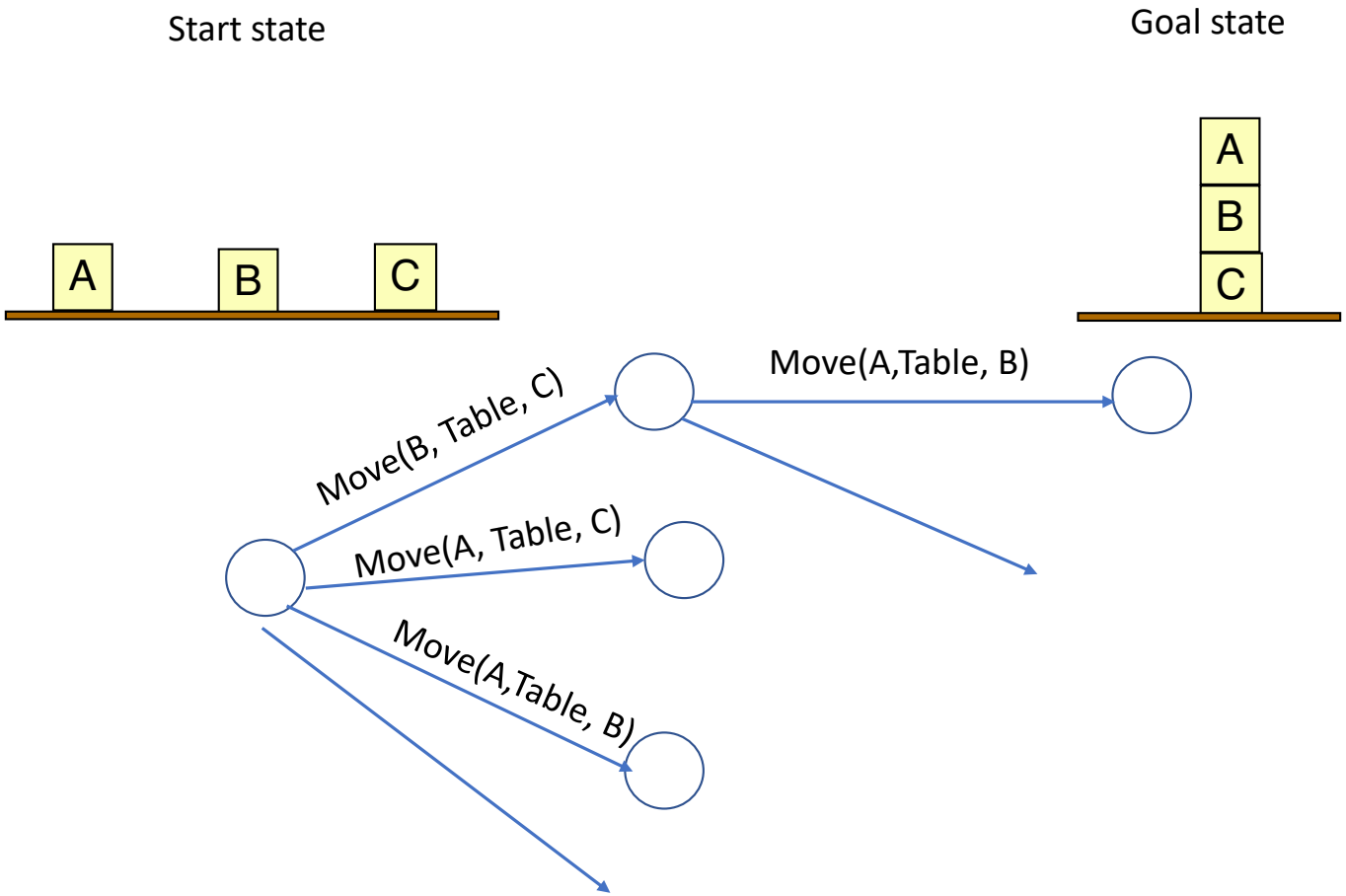


Example: Forward Chaining in Block World-3

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)



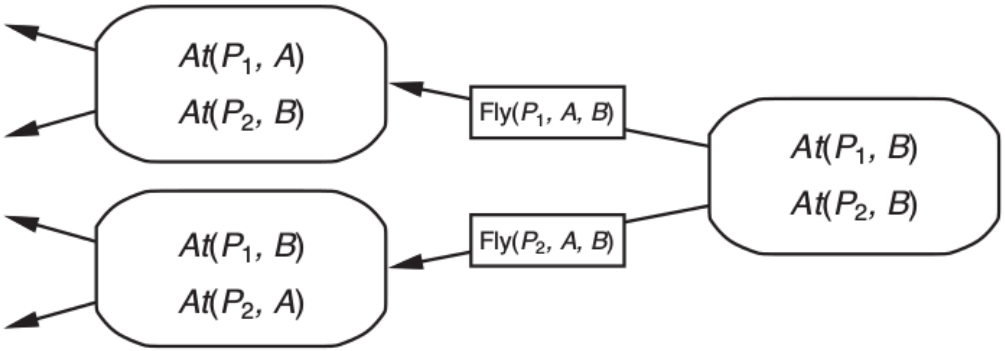
Example: Forward Chaining in Block World-4



Branching Factor of Forward Search

- Forward search can have a very large branching factor
  - E.g., many applicable actions that don't progress toward goal
- Why this is bad: The search algorithms we saw, e.g., in chapter 3 can waste time trying lots of irrelevant actions
- Need good (domain-specific) heuristic and/or pruning procedure

Regression (backward) planning



Algorithm:

1. Pick an action that satisfies (some of) the goal propositions.
2. Make a new goal by:
   - Removing the goal conditions satisfied by the action
   - Adding the preconditions of this action
   - Keeping any unsolved goal propositions
3. Repeat until the goal set is satisfied by the start state

Backward Search

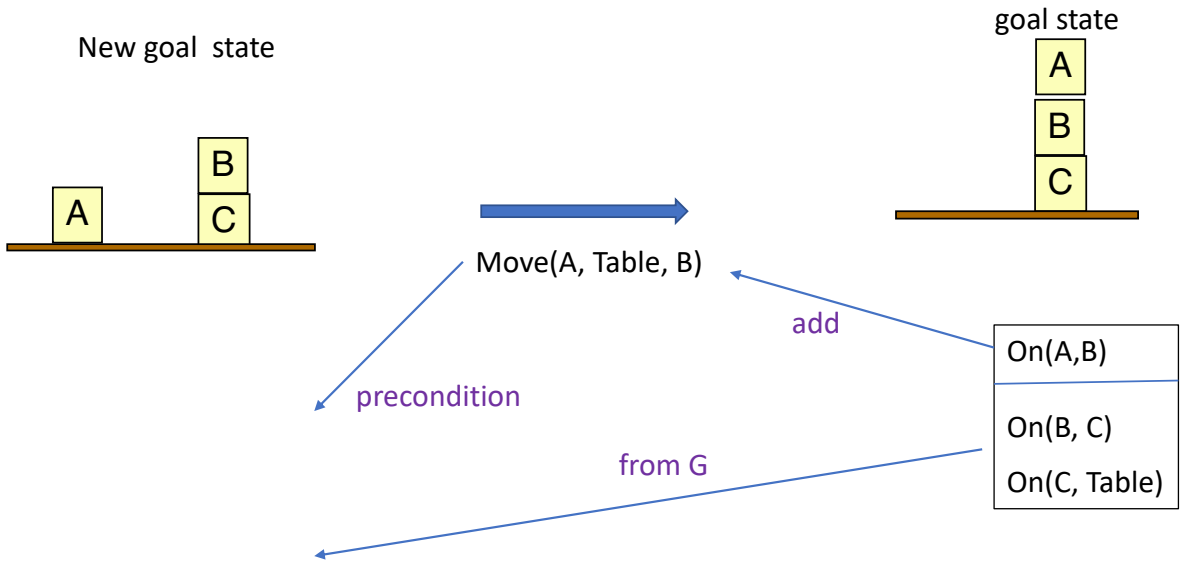
- For forward search, we started at the initial state and computed state transitions
  - new state = γ(s,a)
- For backward search, we start at the goal and compute inverse state transitions
  - new set of subgoals = γ^{-1}(g,a)

Backward Search

- For forward search, we started at the initial state and computed state transitions
  - new state $= \gamma$(s,a)
- For backward search, we start at the goal and compute inverse state transitions
  - new set of subgoals $= \gamma^{-1}$(g,a)
- To define γ^{-1}(g,a), must first define **relevance**:
  An action a is relevant for a goal g if
  - a makes at least one of g's literals true
    $g \cap \text{effects}(a) \neq \Phi$
  - a does not make any of g's literals false
    $g^{+} \cap \text{effects}^{-}(a) = \Phi$ and
    $g^{-} \cap \text{effects}^{+}(a) = \Phi$
    where g^{-} = negative fluents in g

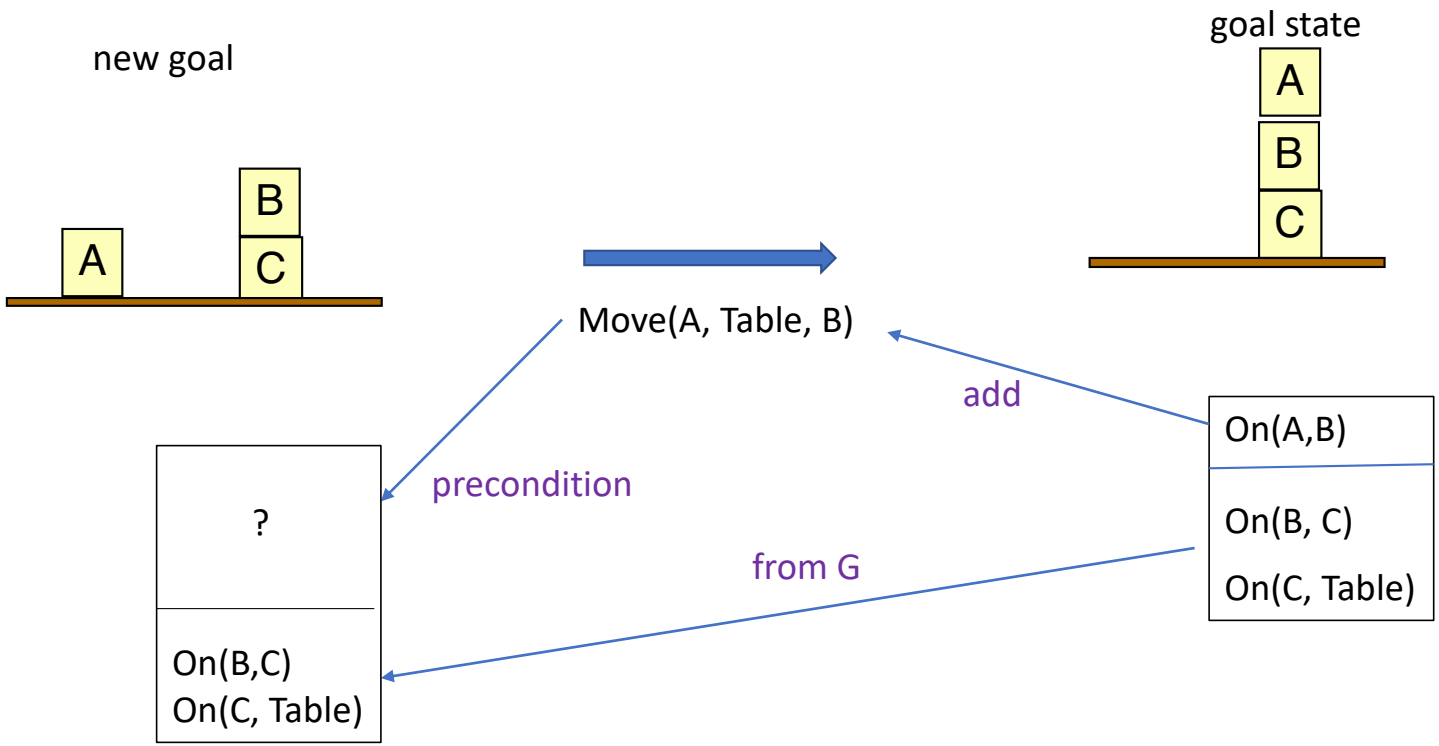
Example: Backward Chaining in Block World

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)



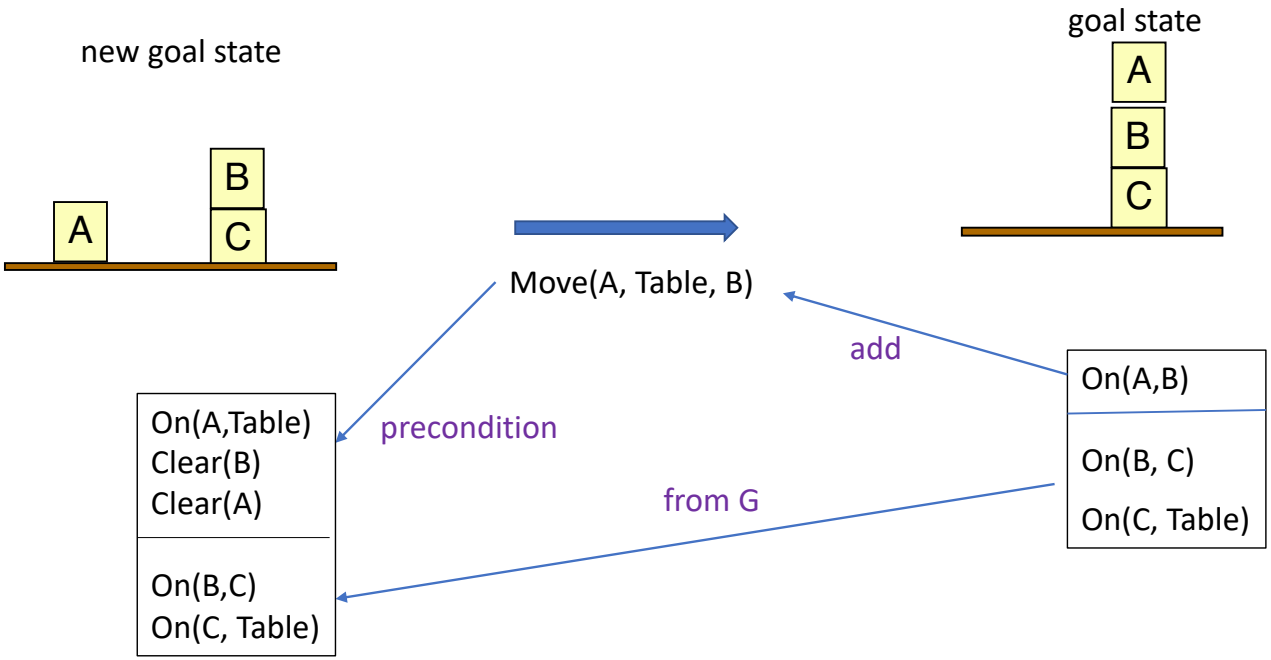
Example: Backward Chaining in Block World-2

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)

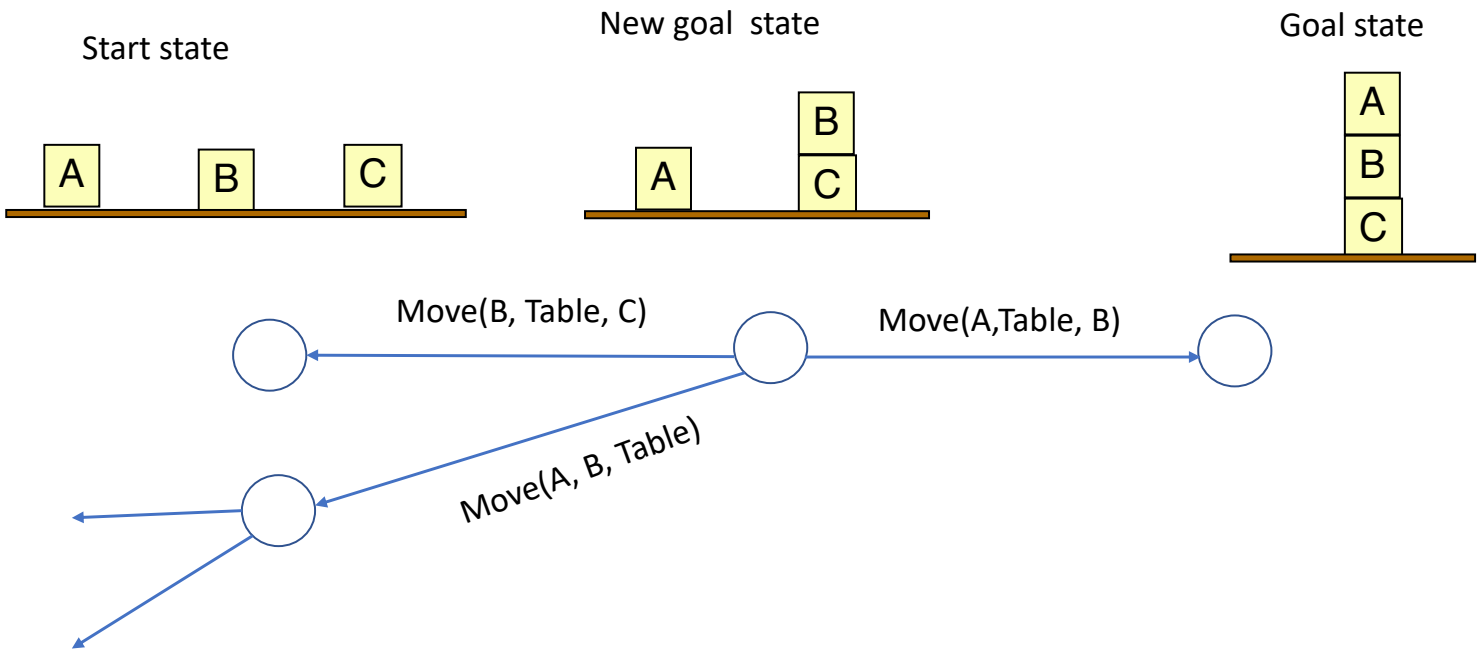


Example: Backward Chaining in Block World-3

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)



Example: Backward Chaining in Block World-4



Efficiency of Backward Search

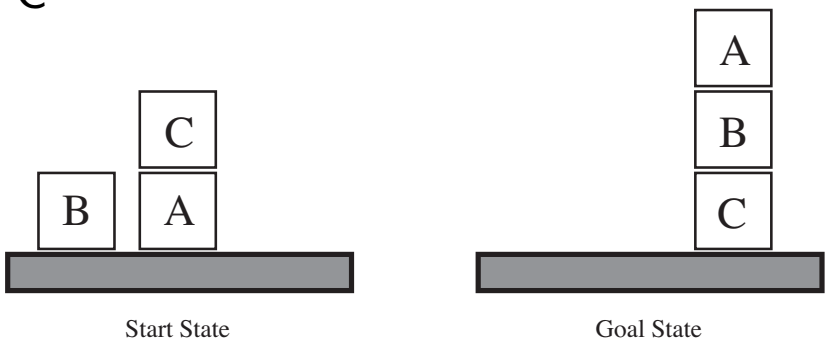
- Not guided regarding which subgoal to pursue at a moment, hence inefficient
- Note that the order in which we try to achieve these subgoals matters!
- We'll next talk about the subgoal dependencies and planning algorithms that handle this.

Disadvantages of State-space search

- Forward and Backward planning works with strictly linear sequences of actions
- No mechanism for least commitment in terms of action ordering

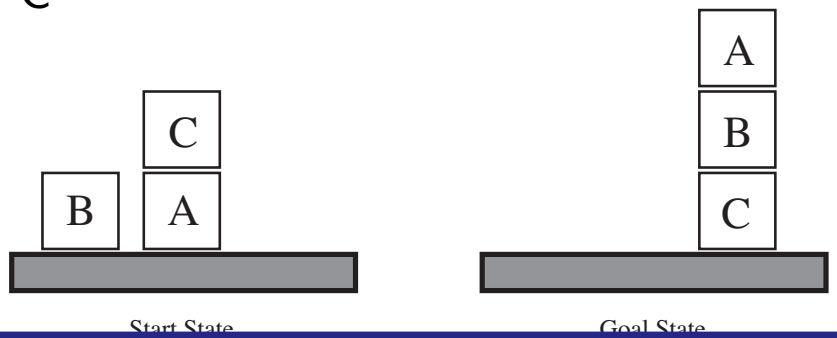
Subgoal Dependency and Sussman Anomaly

- Forward and Backward Planning in state-space are "Total ordering" algorithms.
- Total order planners typically separate the goal (e.g., in the Block World : stack A atop B atop C) into subgoals, such as:
    1. get A atop B
    2. get B atop C



Subgoal Dependency and Sussman Anomaly

- Forward and Backward Planning in state-space are "Total ordering" algorithms.
- Total order planners typically separate the goal (e.g., in the Block World : stack A atop B atop C) into subgoals, such as:
  1. get A atop B
  2. get B atop C

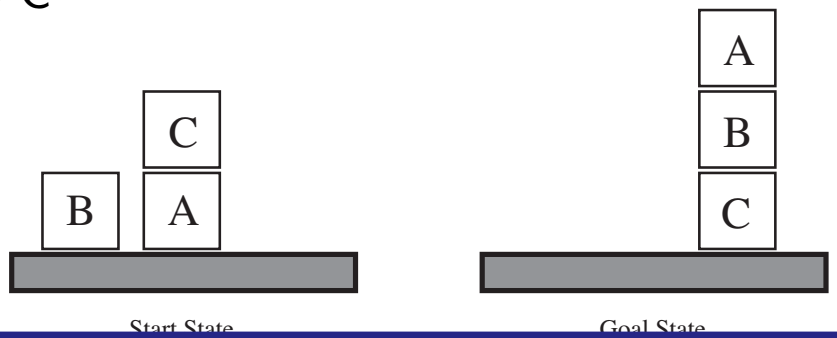


Start State Goal State

Accomplish subgoal 1 first by removing C from A and moving A atop B ... but then we cannot accomplish subgoal 2 without undoing subgoal.

Subgoal Dependency and Sussman Anomaly

- Forward and Backward Planning in state-space are "Total ordering" algorithms.
- Total order planners typically separate the goal (e.g., in the Block World : stack A atop B atop C) into subgoals, such as:
  1. get A atop B
  2. get B atop C



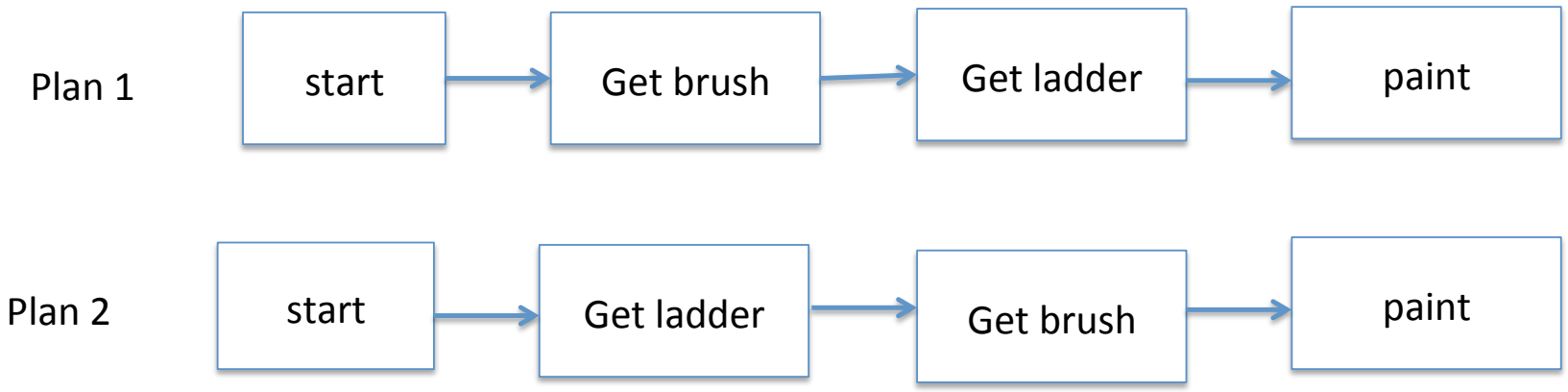
Accomplish subgoal 1 first by removing C from A and moving A atop B ... but then we cannot accomplish subgoal 2 without undoing subgoal.

Accomplish subgoal 2 first ... then subgoal 1 cannot be achieved without undoing subgoal 2.

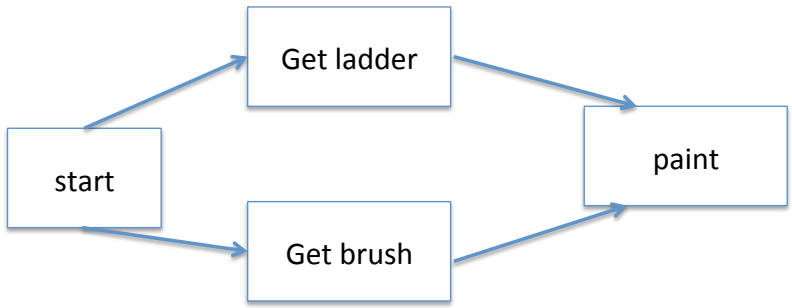
Either way of ordering, subgoals cause clobbering.

Total vs Partial Order Planning

Total order: Plan is always a strict sequence of actions



Partial order: Plan steps may be unordered



Total vs Partial Order Planning

- The planning methods so far generated total-order plans - strict order of execution of actions
- However, in some cases, the subgoals/subproblems are independent
- Hence some actions may be executed in parallel (e.g., get brush and get ladder)
- Partial order planning on **plan space** creates more optimal plans

Partial Order Planning

Search in plan space and use least commitment to ordering whenever possible

In state-space search:

- Search space is a set of states of the world
- Actions cause transitions between states
- Plan is a path through state space

Partial Order Planning

Search in plan space and use least commitment to ordering whenever possible

In state-space search:

- Search space is a set of states of the world
- Actions cause transitions between states
- Plan is a path through state space

In plan-space search:

- Search space is a set of partially ordered plans
- Plan operators cause transitions
- Goal is a legal plan

Partial Order Planning Process

- Start with an empty plan consisting of
  - *Start* step with the initial state description as its effect
  - *Finish* step with the goal description as its precondition

Partial Order Planning Process

- Start with an empty plan consisting of
  - *Start* step with the initial state description as its effect
  - *Finish* step with the goal description as its precondition
- Proceed by the operations:

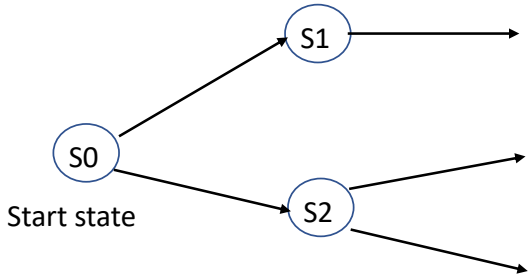
  Plan operators: add actions, add causal links, specify orderings, bind variables
  - adding actions to achieve preconditions
  - adding causal links from an existing action to achieve preconditions
  - order on action w.r.t. another to remove possible conflicts/threats

Partial Order Planning Process

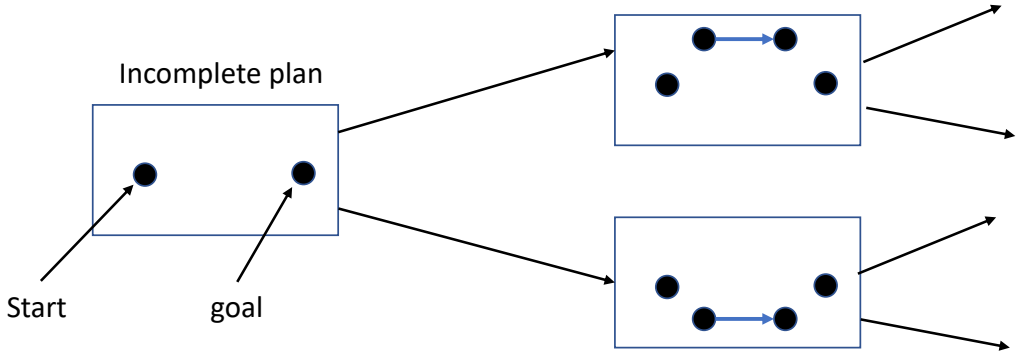
- Start with an empty plan consisting of
  - *Start* step with the initial state description as its effect
  - *Finish* step with the goal description as its precondition
- Proceed by the operations:
  Plan operators: add actions, add causal links, specify orderings, bind variables
  - adding actions to achieve preconditions
  - adding causal links from an existing action to achieve preconditions
  - order on action w.r.t. another to remove possible conflicts/threats
- Gradually move from incomplete/vague plans to complete, correct plans
- Backtrack if an open condition is unachievable or if a conflict is unresolvable
- 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

State-space versus plan-space planning

Planning in state space

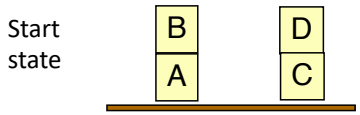


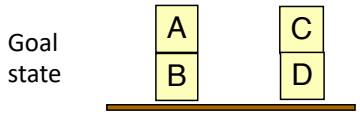
Planning in plan space



POP Example

In this example on Partial order Planning (POP) the subproblems/subgoals are independent



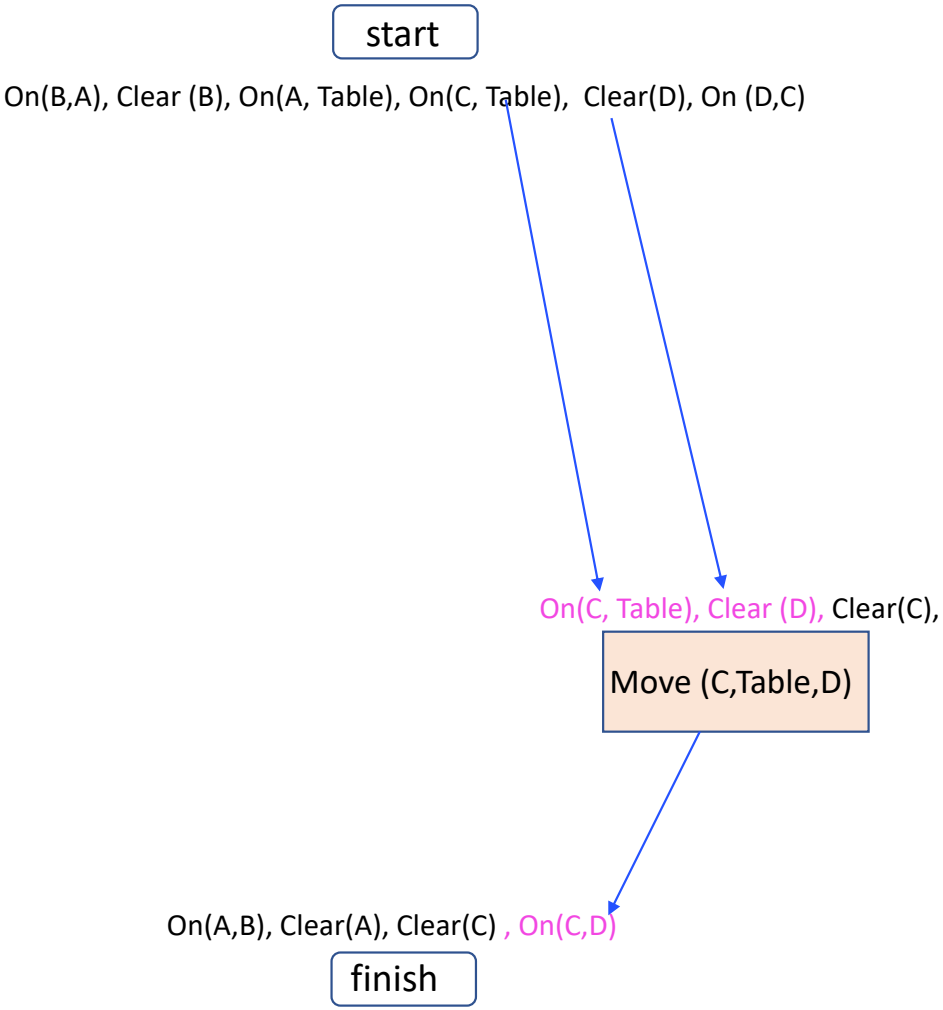


Assume we have 2 actions:

$Action(Move(b, x, y)$
PRECOND : $On(b, x) \wedge Clear(b) \wedge Clear(y)$
EFFECT : $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$

$Action(Move\text{-}to\text{-}table(b, x)$
PRECOND : $On(b, x) \wedge Clear(b)$
EFFECT : $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

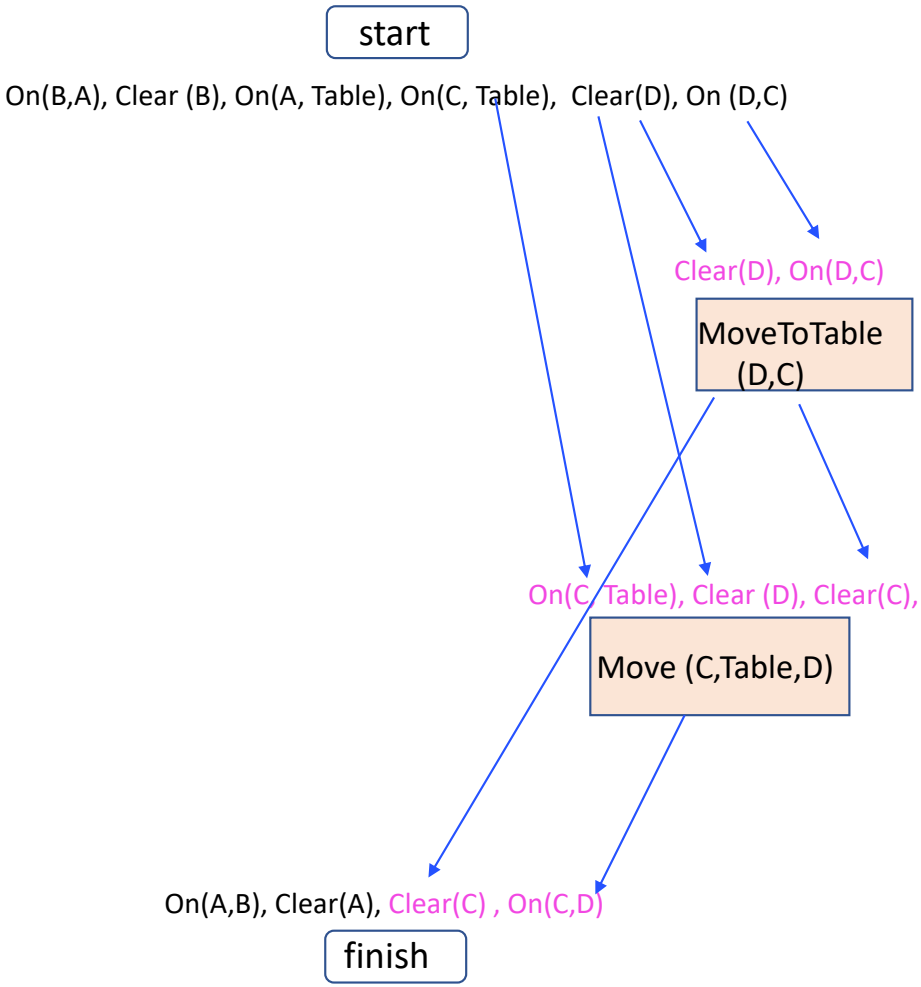
Example - 2



$Action(Move(b, x, y)$
PRECOND : $On(b, x) \wedge Clear(b) \wedge Clear(y)$
EFFECT : $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y$

$Action(Move\text{-}to\text{-}table(b, x)$
PRECOND : $On(b, x) \wedge Clear(b)$
EFFECT : $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

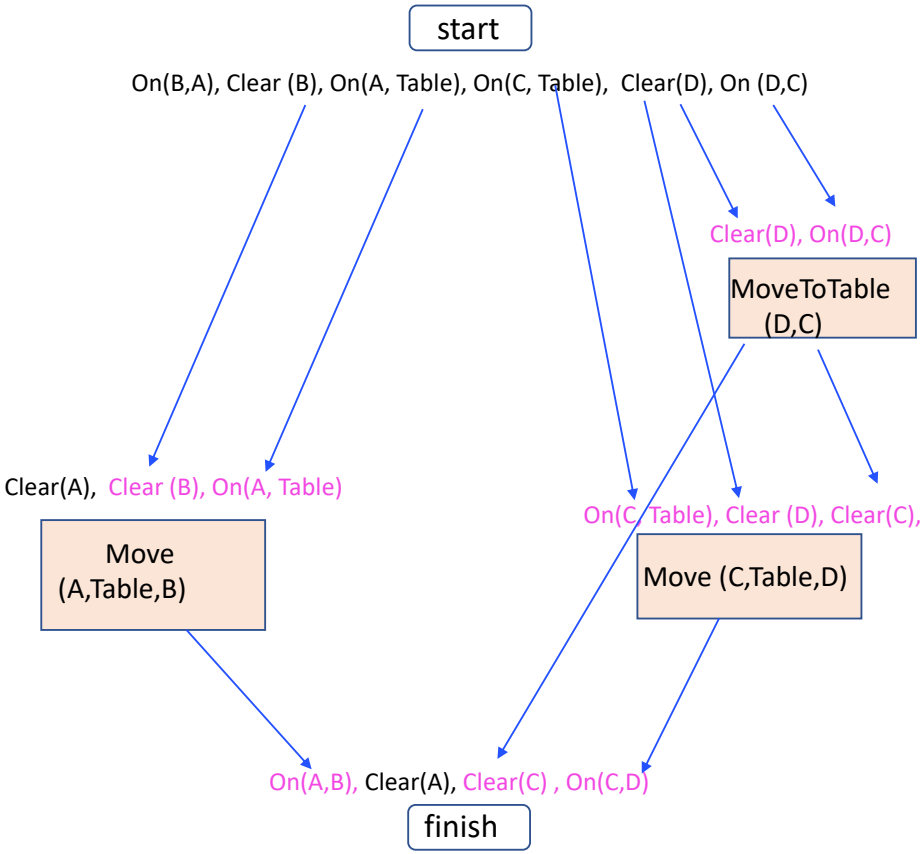
Example - 3



$Action(Move(b, x, y)$
PRECOND : $On(b, x) \wedge Clear(b) \wedge Clear(y)$
EFFECT : $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y$

$Action(Move\text{-}to\text{-}table(b, x)$
PRECOND : $On(b, x) \wedge Clear(b)$
EFFECT : $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

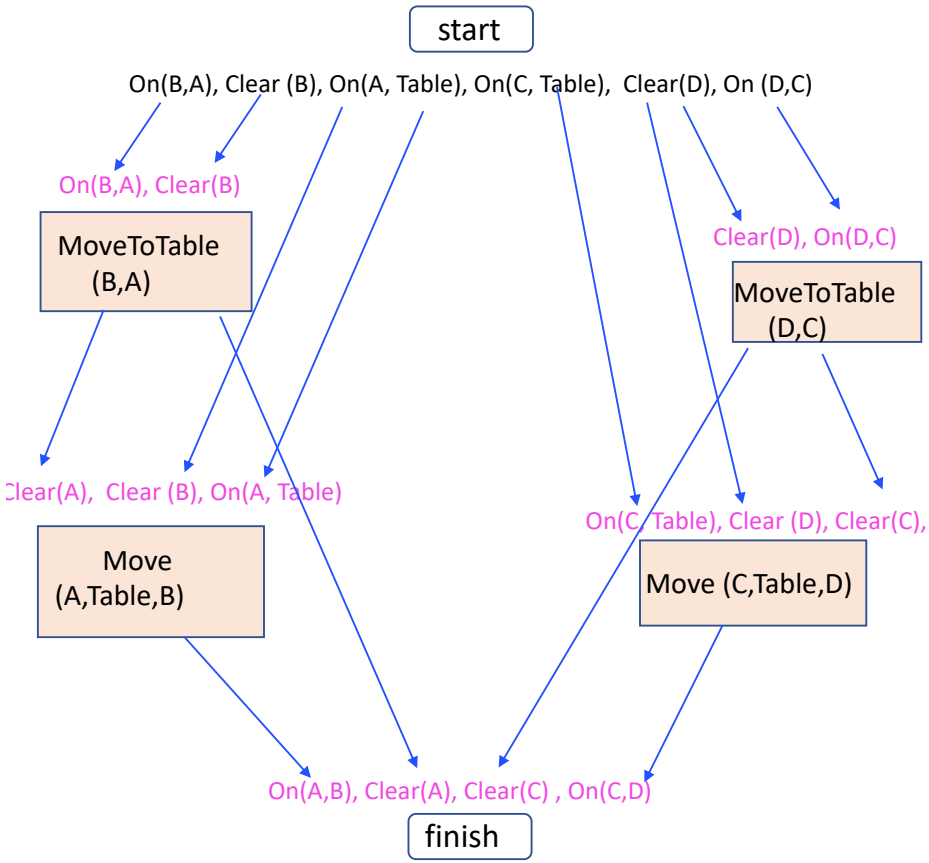
Example - 4



$Action(Move(b, x, y)$
PRECOND : $On(b, x) \wedge Clear(b) \wedge Clear(y)$
EFFECT : $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y$

$Action(Move\text{-}to\text{-}table(b, x)$
PRECOND : $On(b, x) \wedge Clear(b)$
EFFECT : $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

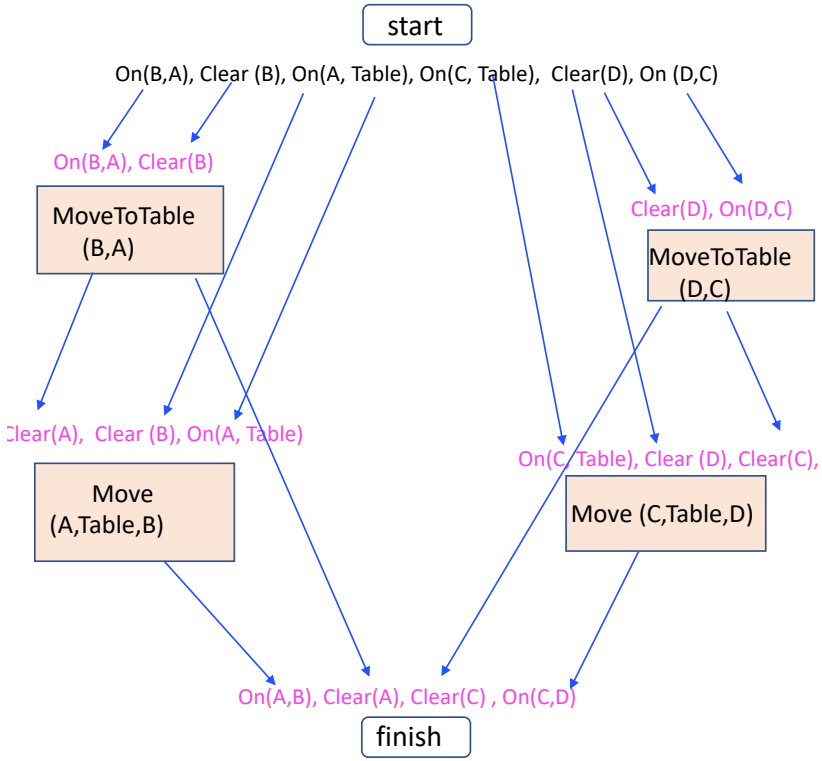
Example - 5



$Action(Move(b, x, y)$
PRECOND : $On(b, x) \wedge Clear(b) \wedge Clear(y)$
EFFECT : $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y$

$Action(Move\text{-}to\text{-}table(b, x)$
PRECOND : $On(b, x) \wedge Clear(b)$
EFFECT : $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

Example - 6



- Two subgoals, found a plan for each, P1 and P2
  - P1: MovetoTable(B,A) + Move(A, Table, B)
  - P2: MoveToTable (D,C) + Move(C,Table, D)
- Since the subgoals are independent, i.e., like two separate problem instances:
  - Any total order plan consistent with this partial order will work.
  - E.g.: MoveToTable(B,A), MoveToTable(D,C), Move(C, Table, D), Move(A, Table, B)

Pros and Cons of Partial Order Planning

Advantages:

- Plan steps may be executed unordered
- Handles concurrent plans
- Least commitment can lead to shorter search times
- Sound and complete
- Typically produces the optimal plan

Disadvantage:

Complex plan operators lead to high cost for generating actions

A solution may be Planning Graphs but it is not included in the syllabus this year.

Heuristics for State-Space Search

- How to find an admissible heuristic estimate?
  - Distance from a state to the goal?
  - Look at the effects of the actions and guess how many actions are needed
  - NP-hard

Heuristics for State-Space Search

- How to find an admissible heuristic estimate?
  - Distance from a state to the goal?
  - Look at the effects of the actions and guess how many actions are needed
  - NP-hard
- Relaxed problem

Relaxed Problem

- Delete preconditions heuristic
  - Example : 8- tiles problem
    - Action(Slide(t, s1,s2)
      PRECOND: On(t,s1) $\wedge$ Tile(t) $\wedge$Blank(s2) $\wedge$ Adjacent(s1,s2)
      EFFECT: On(t,s2) $\wedge$ Tile(t) $\wedge$Blank(s1) $\wedge \neg$ On(t,s1) $\wedge \neg$ Blank(s2))
    - remove the preconditions Blank(s2)$\wedge$ Adjacent(s1,s2) - we get the nr-of-misplaced tiles heuristic
    - remove the precondition Blank(s2) - we get the Manhattan distance heuristic

Relaxed Problem

- Delete preconditions heuristic
  - Example : 8- tiles problem
    - Action(Slide(t, s1,s2)
      PRECOND: On(t,s1) $\wedge$ Tile(t) $\wedge$Blank(s2) $\wedge$ Adjacent(s1,s2)
      EFFECT: On(t,s2) $\wedge$ Tile(t) $\wedge$Blank(s1) $\wedge \neg$ On(t,s1) $\wedge \neg$ Blank(s2))
    - remove the preconditions Blank(s2)$\wedge$ Adjacent(s1,s2) - we get the nr-of-misplaced tiles heuristic
    - remove the precondition Blank(s2) - we get the Manhattan distance heuristic
- Ignore delete-list heuristic
  - If we assume that all goals and preconditions have only positive literal
  - length of the solution will be a good heuristic
  - to find the length, actions must not undo each other.
  - This is possible when the delete-effects of actions are ignored.

Problem Solving vs Planning

Key ideas:

- Factored Representation
  - States/goals represented by sets of sentences
  - Actions represented by logical description of preconditions and effects
  - Planner enabled to make direct connections between states and actions
- Most parts of the world are independent of most other parts of the world
  - Conjunctions can be separated and handled independently (divide-and-conquer)
  - Can handle interdependency between the goals

Planning and acting in Non-classical Environments

- Assumptions so far, in "Classical Planning":
  - the environment is deterministic
  - the agent knows the effects of each action
  - the environment is fully observable
  - the environment is static

Planning and acting in Non-classical Environments

- Assumptions so far, in "Classical Planning":
  - the environment is deterministic
  - the agent knows the effects of each action
  - the environment is fully observable
  - the environment is static
- Here the agent does not need perception:
  - can calculate which state results from any sequence of actions
  - always knows which state it is in.

Planning and acting in Non-classical Environments

- Assumptions so far, in "Classical Planning":
  - the environment is deterministic
  - the agent knows the effects of each action
  - the environment is fully observable
  - the environment is static
- Here the agent does not need perception:
  - can calculate which state results from any sequence of actions
  - always knows which state it is in.
- In the real world, the environment may be uncertain:
  - partially observable and/or nondeterministic environment
  - incorrect information (differences between world and model)

  $\Longrightarrow$ if one of the above assumptions does not hold, use percepts
  - the agent's future actions will depend on future percepts
  - the future percepts cannot be determined in advance

Planning and acting in Non-classical Environments

- Assumptions so far, in "Classical Planning":
  - the environment is deterministic
  - the agent knows the effects of each action
  - the environment is fully observable
  - the environment is static
- Here the agent does not need perception:
  - can calculate which state results from any sequence of actions
  - always knows which state it is in.
- In the real world, the environment may be uncertain:
  - partially observable and/or nondeterministic environment
  - incorrect information (differences between world and model)

  $\Longrightarrow$ if one of the above assumptions does not hold, use percepts
  - the agent's future actions will depend on future percepts
  - the future percepts cannot be determined in advance
- Use percepts:
  - perceive the changes in the world
  - act accordingly
  - adapt plan when necessary

Types of Planning in "non-classical" environments

- Sensorless planning (conformant planning):
  - for environments with no observations
  - constructs sequential plans to be executed without perception
  - find plan that achieves goal in all possible circumstances (regardless of initial state and action effects)

Types of Planning in "non-classical" environments

- Sensorless planning (conformant planning):
  - for environments with no observations
  - constructs sequential plans to be executed without perception
  - find plan that achieves goal in all possible circumstances (regardless of initial state and action effects)
- Conditional planning (contingent planning):
  - for partially-observable and nondeterministic environments
  - constructs a conditional plan with different branches for different contingencies
- Online PLanning - execution monitoring and replanning:
  - for partially-known or evolving environments
  - it monitors the execution process and. replan when necessary

Closed-World Assumption vs Open-World Assumption

- Classical Planning based on Closed-World Assumption
  - states contain only positive fluents
  - we assume that every fluent not mentioned in a state is false
- Sensorless and Partially-observable Planning based on Open-World Assumption
  - states contain both positive and negative fluents
  - if a fluent does not appear in the state, its value is unknown

Closed-World Assumption vs Open-World Assumption

- Classical Planning based on Closed-World Assumption
  - states contain only positive fluents
  - we assume that every fluent not mentioned in a state is false
- Sensorless and Partially-observable Planning based on Open-World Assumption
  - states contain both positive and negative fluents
  - if a fluent does not appear in the state, its value is unknown
- Planning in Belief space
  - A belief state is represented by a logical formula (instead of an explicitly-enumerated set of states)
  - The belief state corresponds exactly to the set of possible worlds that satisfy the formula representing it
- The unknown information can be retrieved via sensing actions (kind of "percept action schema") added to the plan

Example: Colouring Problem in Partially observable environment

Given a chair and a table, the goal is to have them of the same colour. In the initial state we have two cans of paint, but the colours of the paint and the chair are unknown. Only the table is initially in the agent's field of view.

- Initial state:
  Object(Table) $\wedge$ Object(Chair) $\wedge$ Can(C1) $\wedge$Can(C2) $\wedge$InView(Table)
- Goal state:
  Color(Chair, c) $\wedge$ Color (Table, c)

Example: Colouring Problem in Partially observable environment

Given a chair and a table, the goal is to have them of the same colour. In the initial state we have two cans of paint, but the colours of the paint and the chair are unknown. Only the table is initially in the agent's field of view.

- Initial state:
  Object(Table) $\wedge$ Object(Chair) $\wedge$ Can(C1) $\wedge$Can(C2) $\wedge$InView(Table)
- Goal state:
  Color(Chair, c) $\wedge$ Color (Table, c)
- Actions:
  - RemoveLid (can)
    Precond : Can(can)
    Effect : Open(can)

Example: Colouring Problem in Partially observable environment

Given a chair and a table, the goal is to have them of the same colour. In the initial state we have two cans of paint, but the colours of the paint and the chair are unknown. Only the table is initially in the agent's field of view.

- Initial state:
  Object(Table) $\wedge$ Object(Chair) $\wedge$ Can(C1) $\wedge$Can(C2) $\wedge$InView(Table)
- Goal state:
  Color(Chair, c) $\wedge$ Color (Table, c)
- Actions:
  - RemoveLid (can)
    Precond : Can(can)
    Effect : Open(can)
  - Paint(x, can) - *paint object using paint from a can*
    Precond : Object(x) $\wedge$ Can(can) $\wedge$ Color(can, c) $\wedge$ Open(can)
    Effect : Color(x, c) - *note that c is not in the action's variable list*

Example: Colouring Problem in Partially observable environment

Given a chair and a table, the goal is to have them of the same colour. In the initial state we have two cans of paint, but the colours of the paint and the chair are unknown. Only the table is initially in the agent's field of view.

- Initial state:
  Object(Table) $\wedge$ Object(Chair) $\wedge$ Can(C1) $\wedge$Can(C2) $\wedge$InView(Table)
- Goal state:
  Color(Chair, c) $\wedge$ Color (Table, c)
- Actions:
  - RemoveLid (can)
    Precond : Can(can)
    Effect : Open(can)
  - Paint(x, can) - *paint object using paint from a can*
    Precond : Object(x) $\wedge$ Can(can) $\wedge$ Color(can, c) $\wedge$ Open(can)
    Effect : Color(x, c) - *note that c is not in the action's variable list*
  - LookAt(x) - *causes objects to come into view (one at a time)*
    Precond : InView(y) $\wedge$ (x $\neq$ y)
    Effect : InView(x) $\wedge$ $\neg$ InView(y)

Example: Colouring Problem in Partially observable environment -cont.

- Partially-Observable Problems:
  - need to reason about percepts obtained during action
  - $\Longrightarrow$ PDDL is augment with **percept schemata**
  - E.g., Percept (Color(x, c))
    Precond : Object(x) $\wedge$ InView(x) -meaning: "if an object is in view, then the agent will perceive its color"
  - perception will acquire the truth value of Color(x, c), for every x, c

Example: Colouring Problem in Partially observable environment -cont.

- Partially-Observable Problems:
  - need to reason about percepts obtained during action
  - $\Longrightarrow$ PDDL is augment with **percept schemata**
  - E.g., Percept (Color(x, c))
    Precond : Object(x) $\wedge$ InView(x) -meaning: "if an object is in view, then the agent will perceive its color"
  - perception will acquire the truth value of Color(x, c), for every x, c
  - Percept (Color(can, c))
    Precond : Can(can) $\wedge$ InView(can) $\wedge$ Open(can). - meaning: "if an open can is in view, then perceive the paint of colour "
  - perception will acquire the truth value of Color(can, c), for a concrete can and a colour

Example: Colouring Problem in Partially observable environment -cont.

- Partially-Observable Problems:
  - need to reason about percepts obtained during action
  - $\Longrightarrow$ PDDL is augment with **percept schemata**
  - E.g., Percept (Color(x, c))
    Precond : Object(x) $\wedge$ InView(x) -meaning: "if an object is in view, then the agent will perceive its color"
  - perception will acquire the truth value of Color(x, c), for every x, c
  - Percept (Color(can, c))
    Precond : Can(can) $\wedge$ InView(can) $\wedge$ Open(can). - meaning: "if an open can is in view, then perceive the paint of colour "
  - perception will acquire the truth value of Color(can, c), for a concrete can and a colour
- Fully-Observable Problems:

Example: Colouring Problem in Partially observable environment -cont.

- Partially-Observable Problems:
  - need to reason about percepts obtained during action
  - $\Longrightarrow$ PDDL is augment with **percept schemata**
  - E.g., Percept (Color(x, c))
    Precond : Object(x) $\wedge$ InView(x) -meaning: "if an object is in view, then the agent will perceive its color"
  - perception will acquire the truth value of Color(x, c), for every x, c
  - Percept (Color(can, c))
    Precond : Can(can) $\wedge$ InView(can) $\wedge$ Open(can). - meaning: "if an open can is in view, then perceive the paint of colour "
  - perception will acquire the truth value of Color(can, c), for a concrete can and a colour
- Fully-Observable Problems:
  - Percept schemata with no preconditions for each fluent. Ex:
  - E.g., Percept(Color(x, c))
- Nonobservable/Sensorless planning:

Example: Colouring Problem in Partially observable environment -cont.

- Partially-Observable Problems:
  - need to reason about percepts obtained during action
  - $\Longrightarrow$ PDDL is augment with **percept schemata**
  - E.g., Percept (Color(x, c))
    Precond : Object(x) $\wedge$ InView(x) -meaning: "if an object is in view, then the agent will perceive its color"
  - perception will acquire the truth value of Color(x, c), for every x, c
  - Percept (Color(can, c))
    Precond : Can(can) $\wedge$ InView(can) $\wedge$ Open(can). - meaning: "if an open can is in view, then perceive the paint of colour "
  - perception will acquire the truth value of Color(can, c), for a concrete can and a colour
- Fully-Observable Problems:
  - Percept schemata with no preconditions for each fluent. Ex:
  - E.g., Percept(Color(x, c))
- Nonobservable/Sensorless planning:
  no percept schema

Handling Uncertainty in Colouring problem

- Sensorless planning
  - find plan that achieves goal in all possible circumstances. (regardless of initial state and action effects) for environments with no observations
  - example: "Open any can of paint and apply it to both chair and table"

Handling Uncertainty in Colouring problem

- Sensorless planning
  - find plan that achieves goal in all possible circumstances. (regardless of initial state and action effects) for environments with no observations
  - example: "Open any can of paint and apply it to both chair and table"
- Conditional planning
  - construct conditional plan with different branches for possible contingencies for partially-observable and nondeterministic environments
  - ex: "Sense color of table and chair. if they are the same, then finish, else sense can paint. if color(can) $=$ color(chair) then apply color to other piece, else apply color to both"

Handling Uncertainty in Colouring problem

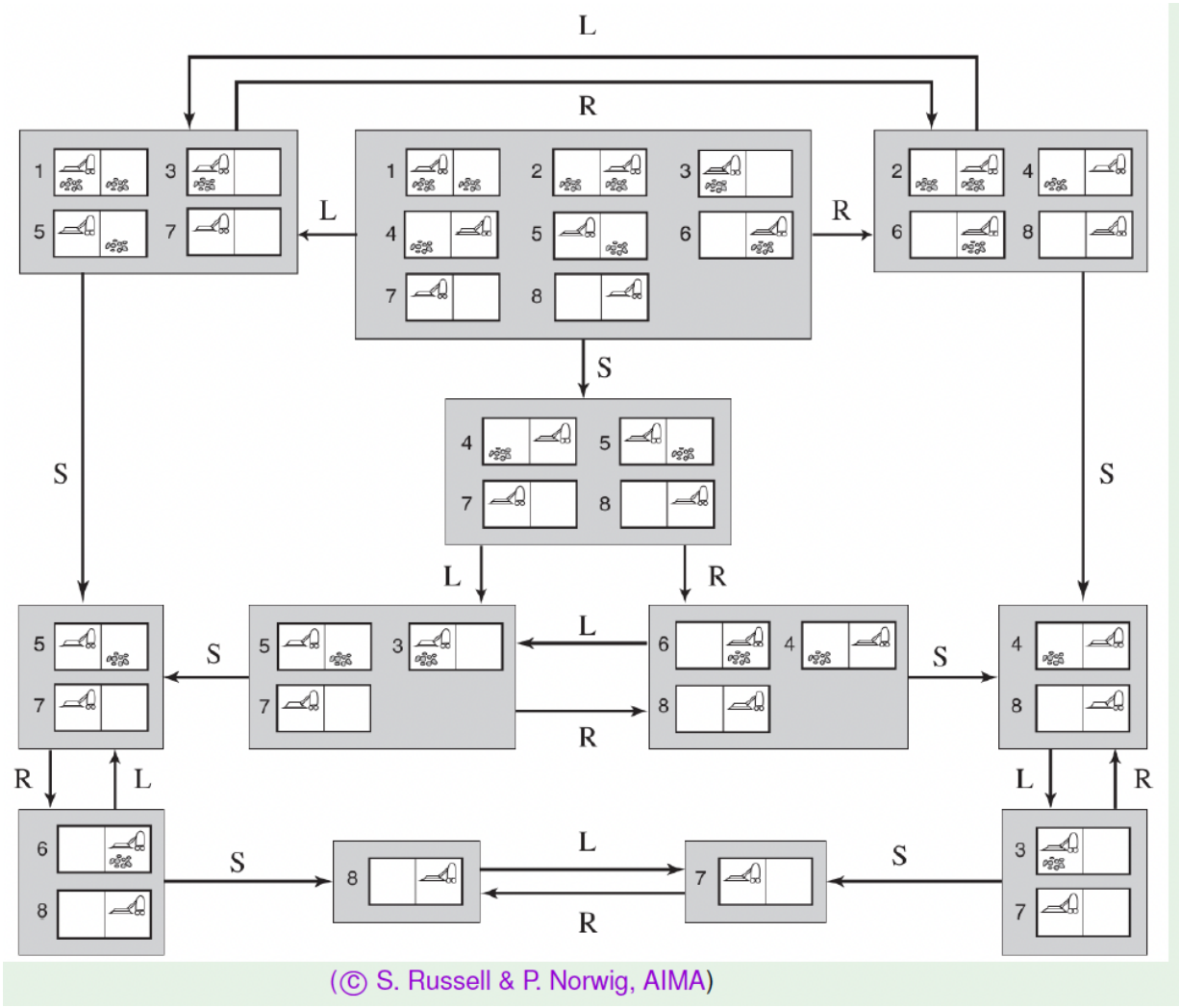
- Sensorless planning
  - find plan that achieves goal in all possible circumstances. (regardless of initial state and action effects) for environments with no observations
  - example: "Open any can of paint and apply it to both chair and table"
- Conditional planning
  - construct conditional plan with different branches for possible contingencies for partially-observable and nondeterministic environments
  - ex: "Sense color of table and chair. if they are the same, then finish, else sense can paint. if color(can) $=$ color(chair) then apply color to other piece, else apply color to both"
- Online planning
  - while constructing plan, judge whether plan requires revision for partially-known or evolving environments
  - ex: Same as conditional, and can fix errors

Conformant Planning for Sensorless agent

- Idea: To solve sensorless problems, the agent searches in the space of belief states rather than in that of physical states fully observable, because the agent knows its own belief space
- Solutions are always sequences of actions (no contingency plan), because percepts are always empty

Recall: Belief-State Problem Formulation

From Chapter 4. for the deterministic sensorless vacuum cleaner



(© S. Russell & P. Norwig, AIMA)

Conformant planning for Sensorless Colouring agent

A sensorless agent cannot have "Percept" in order to acquire more information about environment. It should make a plan that works without information about the colours of the paints in the cans and of the table.

One plan may be:
RemoveLid(Can1),
Paint(Chair, Can1),
Paint(Table, Can1)

Conformant planning for Sensorless Colouring agent

- Open-World Assumption $\Longrightarrow$ a belief state corresponds to the set of possible worlds that satisfy the formula representing it
- All belief states (implicitly) include unchanging facts (invariants) ex: Object(Table) $\wedge$ Object(Chair) $\wedge$ Can(C1) $\wedge$ Can(C2)
- Initial belief state includes facts that are part of the agent's domain knowledge
  - Ex: "objects and cans have colors"
  - $\exists$ Color(x, c), after Skolemization: Color(x, C(x)) in b0.

Conformant planning for Sensorless Colouring agent

- In belief state b, it is possible to apply every action a such that b $\models$ Precond(a)
- e.g., RemoveLid(Can1) applicable in b0 since Can(C1) true in b0
  Remember: RemoveLid (can)
  Precond : Can(can)
  Effect : Open(can)
- Result(b, a) is computed as below:
  - start from b
  - set to false any atom that appears in Del(a) (after unification)
  - set to true any atom that appears in Add(a) (after unification)

Conformant planning for Sensorless Colouring agent

Action(Paint(x, can)
Precond : Object(x) ∧ Can(can) ∧
Color(can, c) ∧ Open(can)
Effect : Color(x, c))

- Start from b0 : Color(x, C(x))
- Apply RemoveLid(Can1) in b0 and obtain:
- b1 : Color(x, C(x)) ∧ Open(Can1)

Conformant planning for Sensorless Colouring agent

Action(Paint(x, can)
Precond : Object(x) ∧ Can(can) ∧
Color(can, c) ∧ Open(can)
Effect : Color(x, c))

- Start from b0 : Color(x, C(x))
- Apply RemoveLid(Can1) in b0 and obtain:
- b1 : Color(x, C(x)) ∧ Open(Can1)
- Apply Paint(Chair, Can1) in b1 using {x=Can1, c=C(Can1)}:
- b2 : Color(x,C(x)) ∧ Open(Can1) ∧ Color(Chair, C(Can1))

Conformant planning for Sensorless Colouring agent

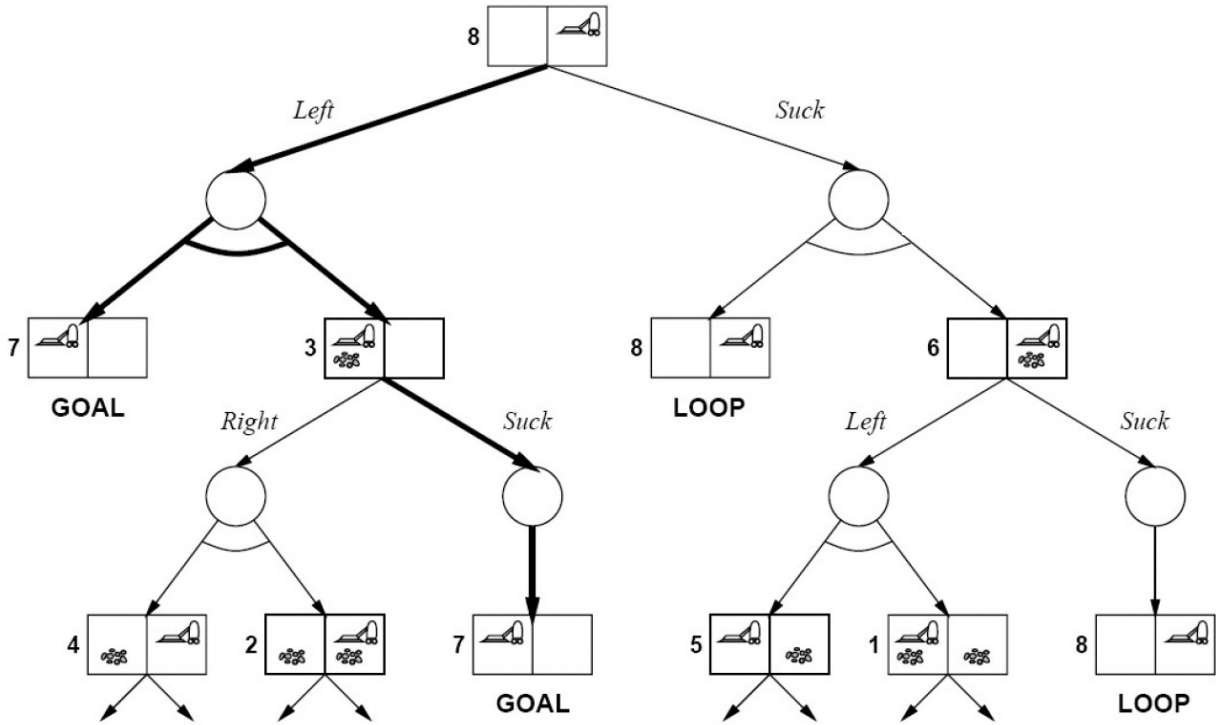
Action(Paint(x, can)
Precond : Object(x) ∧ Can(can) ∧
Color(can, c) ∧ Open(can)
Effect : Color(x, c))

- Start from b0 : Color(x, C(x))
- Apply RemoveLid(Can1) in b0 and obtain:
- b1 : Color(x, C(x)) ∧ Open(Can1)
- Apply Paint(Chair, Can1) in b1 using {x=Can1, c=C(Can1)}:
- b2 : Color(x,C(x)) ∧ Open(Can1) ∧ Color(Chair, C(Can1))
- Apply Paint(Table, Can1) in b2 :
- b3 : Color(x, C(x)) ∧ Open(Can1) ∧ Color(Chair, C(Can1)) ∧ Color(Table, C(Can1))
- b3 Satisfies the goal: b3 $\models$ Color(Table, c) ∧ Color(Chair, c)
- The plan: [RemoveLid(Can1); Paint(Chair, Can1); Paint(Table, Can1)] a valid conformant plan

Planning with nondeterministic actions

- the planning language needs to allow for conditional effects for handling nondeterministic actions
- Conditional effect: when <condition> : <effect >
- We use the vacuum cleaner(VC) example. Left and Right locations.
- Ex: Action(Suck
  EFFECT: when AtL : CleanL $\wedge$ when AtR: CleanR)
- A vacuum cleaner that may dump dirt on the destination square when it moves, but only if that square is clean. Then
  - Action (Left
    PRECOND: AtR
    EFFECT : AtL$\vee$ (AtL $\wedge$ when CleanL:$\neg$ CleanL))

A Contingent Plan for nondeterministic VC



[Left, **if** CleanL **then** [] **else** Suck]

A Contingent Plan for partially observable colouring problem

Given a chair and a table, the goal is to have them of the same colour. In the initial state we have two cans of paint, but the colours of the paint and the chair are unknown.

- LookAt(Table), LookAt(Chair)
- If Colour(Table, c) $\wedge$ Colour(Chair, c) then NoOp
- Else RemoveLid(Can1), LookAt(Can1), RemoveLid(Can2), LookAt(Can2)
- If Colour(Table, c) $\wedge$ (Colour(can, c), then Paint(Chair, can)
- Else If Colour(Chair, c) $\wedge$ (Colour(can, c), then. Paint(Table, can)
- Else Paint(Chair, Can1), Paint(Table, Can1)

Some (or parts of) of these slides are borrowed from José Luis Ambite, Roberto Sebastiani, Milos Hauskrecht, and Daisy Tang.

Summary