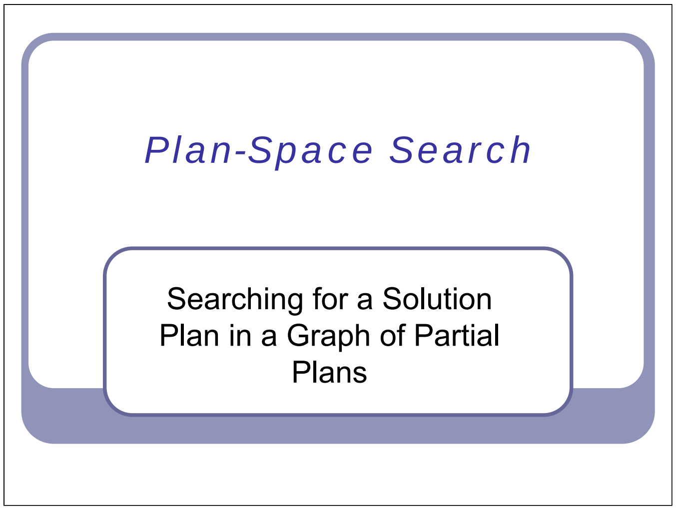
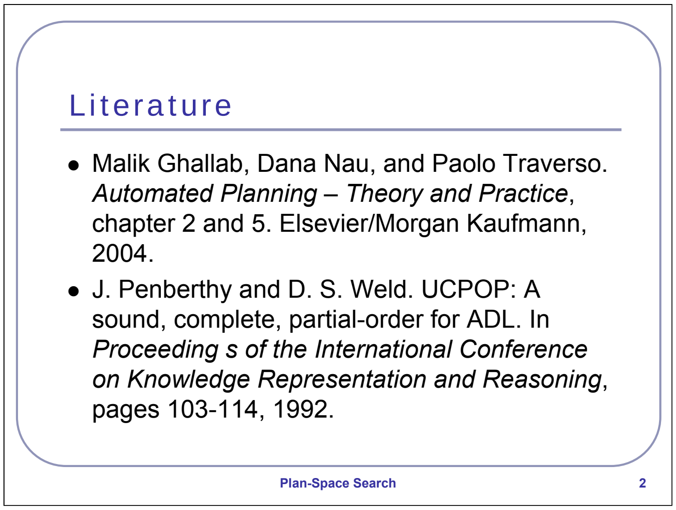


Plan-Space Search

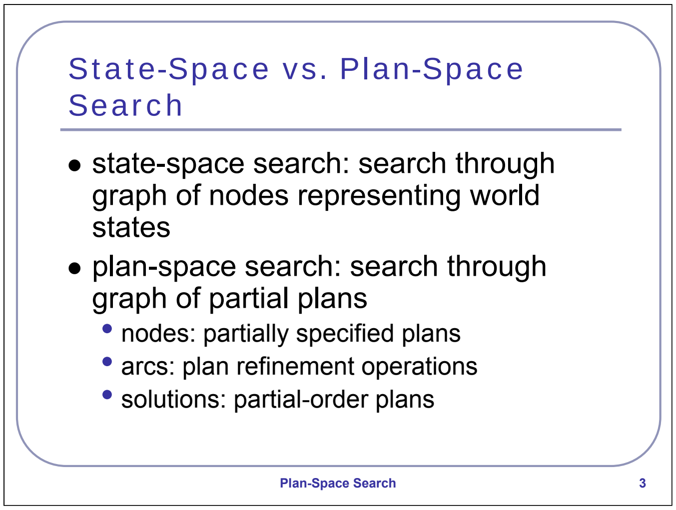
•**Searching for a Solution Plan in a Graph of Partial Plans**



Literature

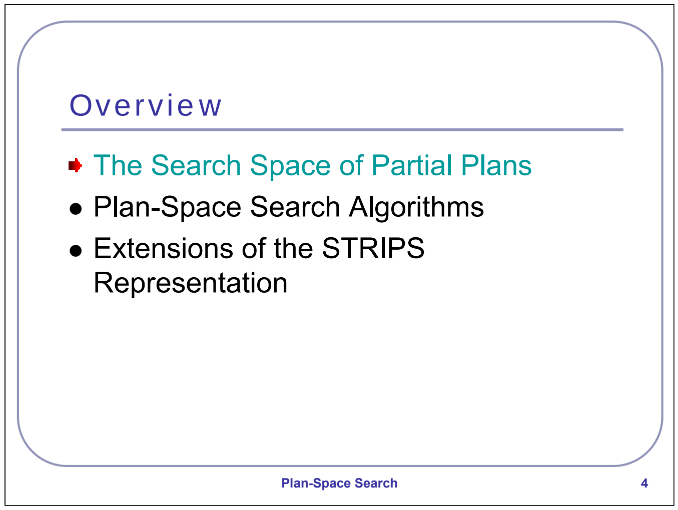
•Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning – Theory and Practice*, chapter 2 and 5. Elsevier/Morgan Kaufmann, 2004.

•J. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial-order for ADL. In *Proceeding s of the International Conference on Knowledge Representation and Reasoning*, pages 103-114, 1992.



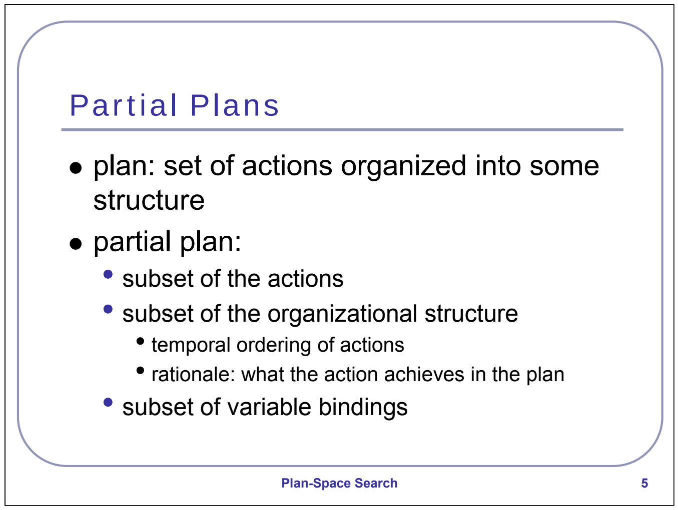
State-Space vs. Plan-Space Search

- **state-space search: search through graph of nodes representing world states**
  - search space directly corresponds to graph representation of state-transition system
- **plan-space search: search through graph of partial plans**
  - **nodes: partially specified plans**
  - **arcs: plan refinement operations**
    - least commitment principle: do not add constraints to the plan that are not strictly needed
  - **solutions: partial-order plans**
    - partial-order plan: set of actions + set of orderings; not necessarily total order
    - state-space algorithms also maintain partial plan – but always in total order



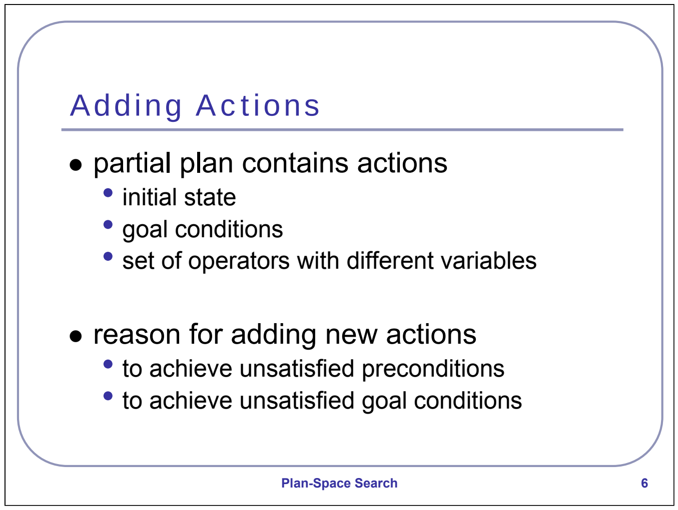
Overview

- **The Search Space of Partial Plans**
  - now: defining the search space: partial plans + plan refinement operations
- **Plan-Space Search Algorithms**
- **Extensions of the STRIPS Representation**



Partial Plans

- **plan: set of actions organized into some structure**
  - organization e.g. sequence
- **partial plan:**
  - **subset of the actions**
  - **subset of the organizational structure**
    - **temporal ordering of actions**
    - **rationale: what the action achieves in the plan**
    - refers only to subset of actions
  - **subset of variable bindings**
- plan refinement operators accordingly: add actions, add ordering constraints, add causal links, add variable bindings



Adding Actions

•partial plan contains actions

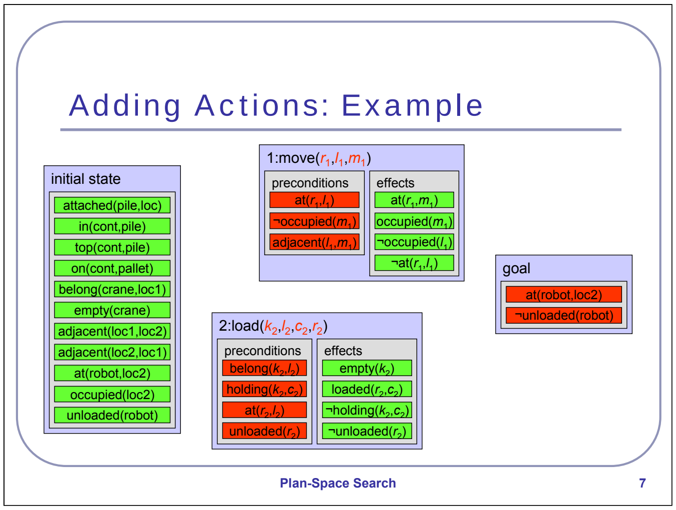
- **•initial state**
- **•goal conditions**
- •can be represented as two actions with only effects or preconditions
- **•set of operators with different variables**

•least commitment principle: introduce actions only for a reason

•reason for adding new actions

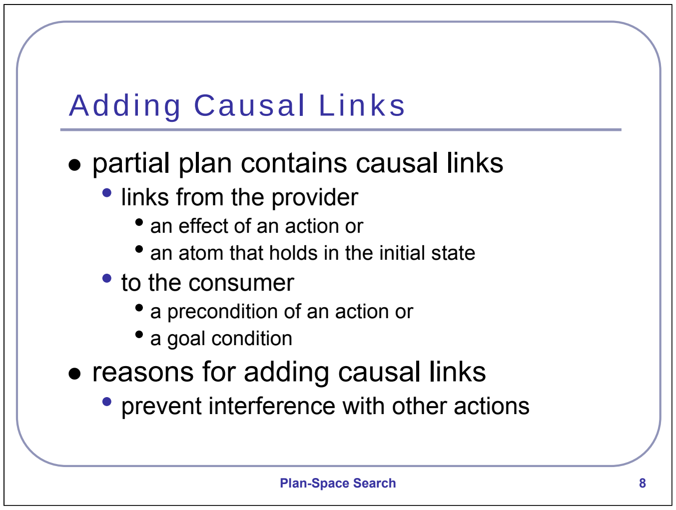
- **•to achieve unsatisfied preconditions**
- **•to achieve unsatisfied goal conditions**

•note: new actions can be added anywhere in the current partial plan



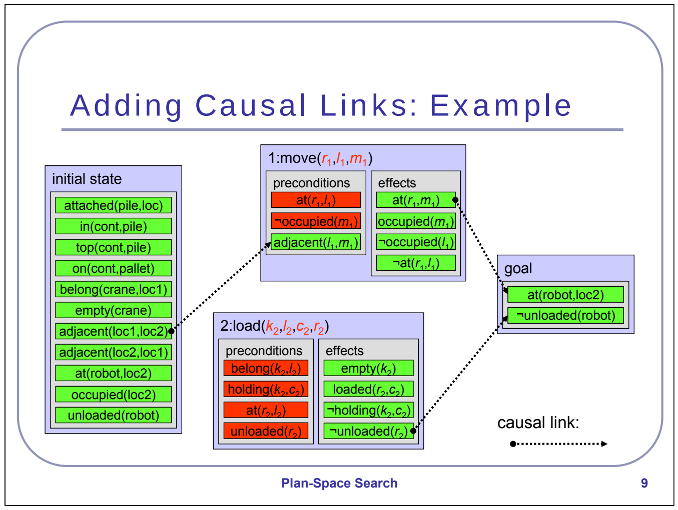
Adding Actions: Example

- empty plan:
    - initial state: all initially satisfied conditions (green)
    - goal: conditions that need to be satisfied (red)
- add operator: 1:move(r_1,l_1,m_1)
    - number (1) to provide unique reference to this operator instance
    - also used as variable index for unique variables
    - least commitment principle: choose values for variables only when necessary
- add operator:2:load(k_2,l_2,c_2,r_2)



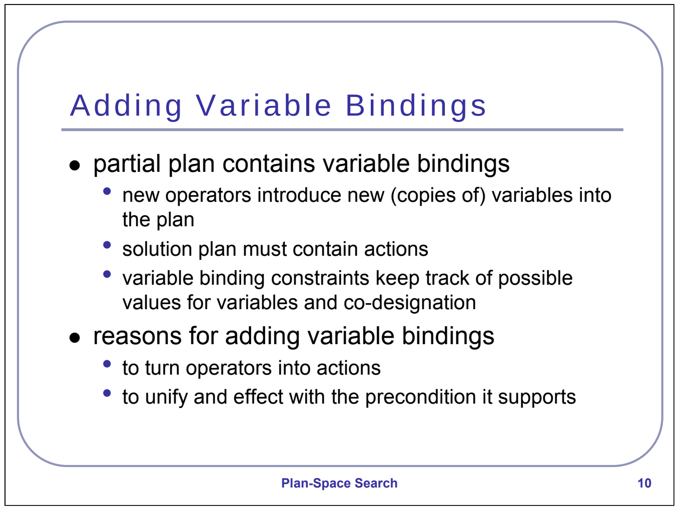
Adding Causal Links

- **partial plan contains causal links**
    - **links from the provider**
        - **an effect of an action or**
        - **an atom that holds in the initial state**
    - **to the consumer**
        - **a precondition of an action or**
        - **a goal condition**
    - causal link implies ordering constraint
        - but: provider need not come directly before consumer
- **reasons for adding causal links**
    - **prevent interference with other actions**
    - keeping track of rationale: any action inserted between provider and consumer must not clobber conditions in causal link
    - preconditions without a causal link pointing to them are open sub-gaols



Adding Causal Links: Example

- add link from 1:move to goal
    - changes colour of goal to green – now satisfied
- add link from 2:load to goal
- add link from initial state to 1:move



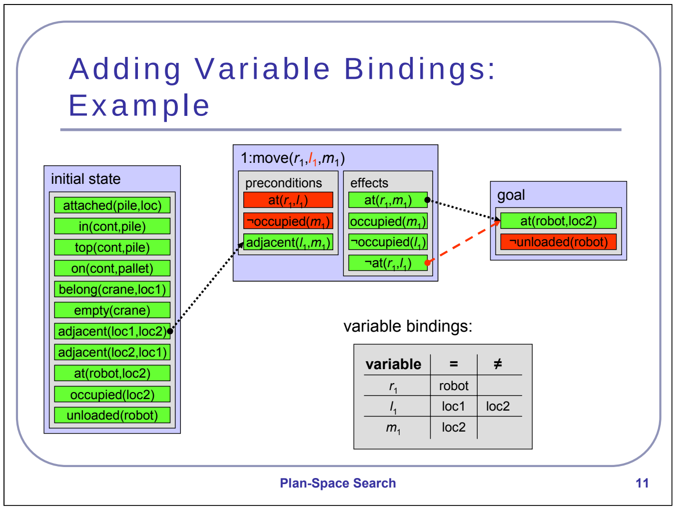
Adding Variable Bindings

•partial plan contains variable bindings

- **•new operators introduce new (copies of) variables into the plan**
  - •each copy of an operator has its own set of variables that are different from variables in other operators instances
- **•solution plan must contain actions**
- **•variable binding constraints keep track of possible values for variables and co-designation**
- •convention (here): give number to operator instances to distinguish them; let variables have index of operator they belong to least commitment principle:
  - •least commitment principle: add only necessary variable binding constraints

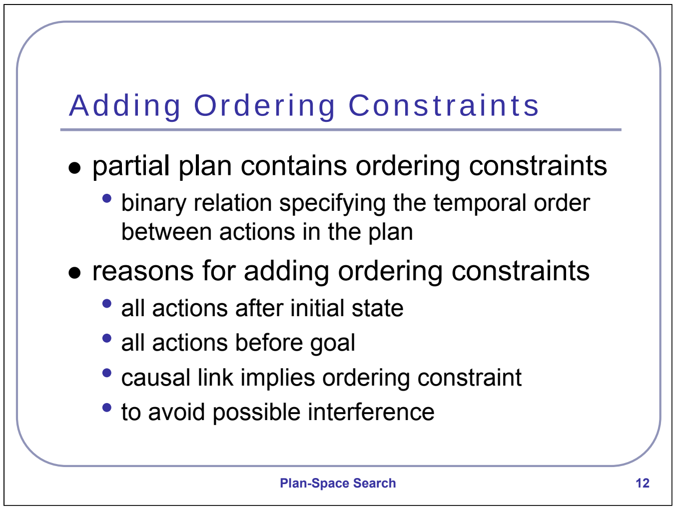
•reasons for adding variable bindings

- **•to turn operators into actions**
- **•to unify and effect with the precondition it supports**



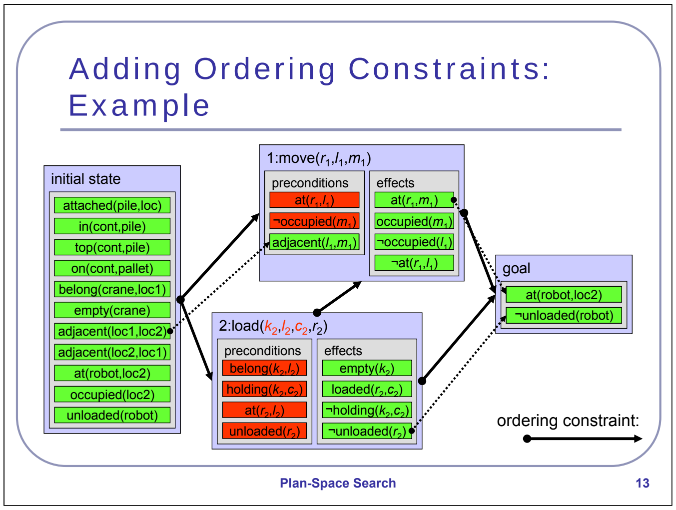
Adding Variable Bindings: Example

- bind variables due to causal link:
  - bind r_1 to robot
  - bind m_1 to loc2
  - note: variables in operator no longer red to indicate they are bound
- clobbering: move may also destroy goal condition
- introduce variable inequality: $l_1 \neq$ loc2
- clobbering now impossible
- introduce causal link from initial state
- bind l_1 to loc1
  - note consistency with inequality



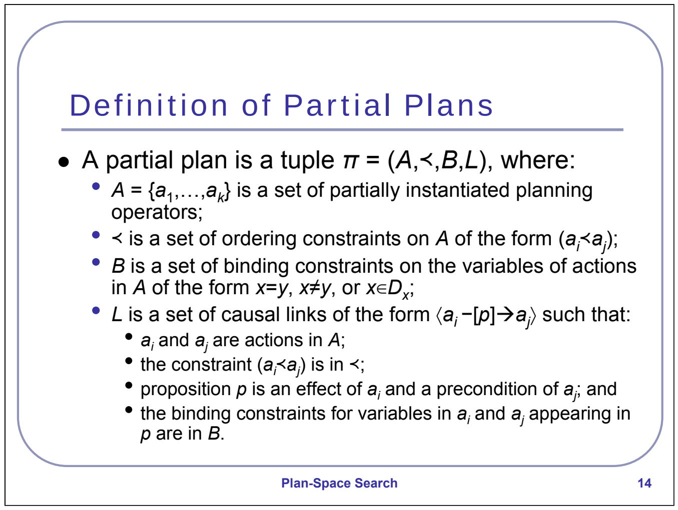
Adding Ordering Constraints

- **partial plan contains ordering constraints**
  - **binary relation specifying the temporal order between actions in the plan**
  - temporal relation: qualitative, not quantitative (at this stage)
- **reasons for adding ordering constraints**
  - **all actions after initial state**
  - **all actions before goal**
  - **causal link implies ordering constraint**
  - **to avoid possible interference**
    - interference can be avoided by ordering the potentially interfering action before the provider or after the consumer of a causal link
    - least commitment principle: introduce ordering constraints only if necessary
- result: solution plan not necessarily totally ordered



Adding Ordering Constraints: Example

- ordering constraints
    - due to causal links
    - also: all actions before goal
- ordering: all actions after initial state
- orderings may occur between actions



Definition of Partial Plans

- **A partial plan is a tuple $\pi = (A,\prec,B,L)$, where:**
  - **$A = \{a_1,\ldots,a_k\}$ is a set of partially instantiated planning operators;**
  - **$\prec$ is a set of ordering constraints on A of the form $(a_i \prec a_j)$;**
  - **B is a set of binding constraints on the variables of actions in A of the form $x=y$, $x \neq y$, or $x \in D_x$;**
  - **L is a set of causal links of the form $\langle a_i -[p] \rightarrow a_j \rangle$ such that:**
    - **a_i and a_j are actions in A;**
    - **the constraint $(a_i \prec a_j)$ is in $\prec$;**
    - **proposition p is an effect of a_i and a precondition of a_j; and**
    - **the binding constraints for variables in a_i and a_j appearing in p are in B.**
- sub-gaols in a partial plan: preconditions without causal links
- different view: partial plan as set of (sequential) plans
  - those that meet the specified constraints and can be refined to a total order plan by adding constraints
- note: partial plans with two types of additional flexibility:
  - actions only partially ordered and
  - not all variables need to be instantiated

Plan-Space Search: Initial Search State

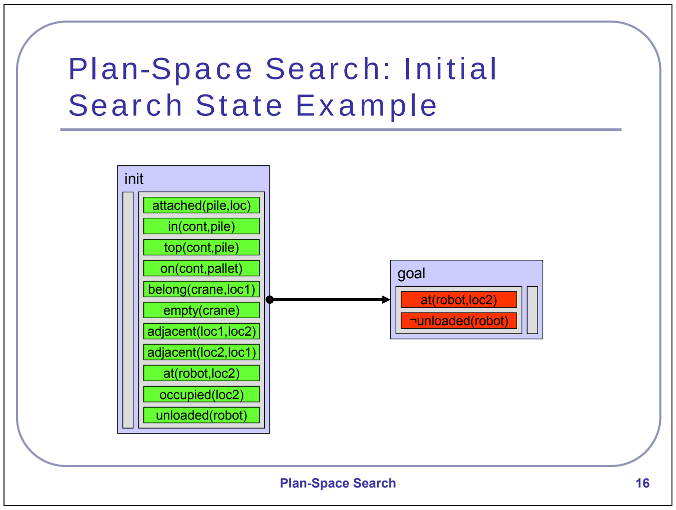
•problem: plan space representation does not maintain states, but need to give initial state and goal description

•represent initial state and goal as dummy actions

- **•init: no preconditions, initial state as effects**
- **•goal: goal conditions as preconditions, no effects**

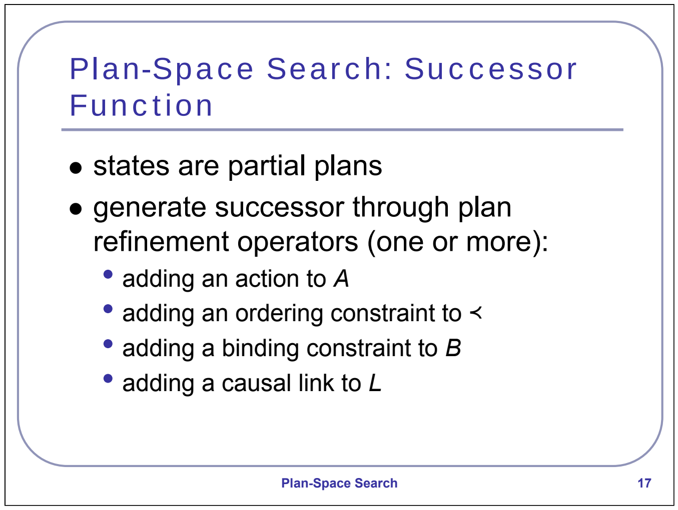
•empty plan π_0 = ({init, goal},{(init≺goal)},{},{}):

- **•two dummy actions init and goal;**
- **•one ordering constraint: init before goal;**
- **•no variable bindings; and**
- **•no causal links.**



Plan-Space Search: Initial Search State Example

•note empty box for preconditions in init and empty box for effects in goal



Plan-Space Search: Successor Function

•states are partial plans

•generate successor through plan refinement operators (one or more):

- •more required to keep partial plans consistent, e.g. adding a causal link implies adding an ordering constraint
- **•adding an action to *A***
- **•adding an ordering constraint to $\prec$**
- **•adding a binding constraint to *B***
- **•adding a causal link to *L***

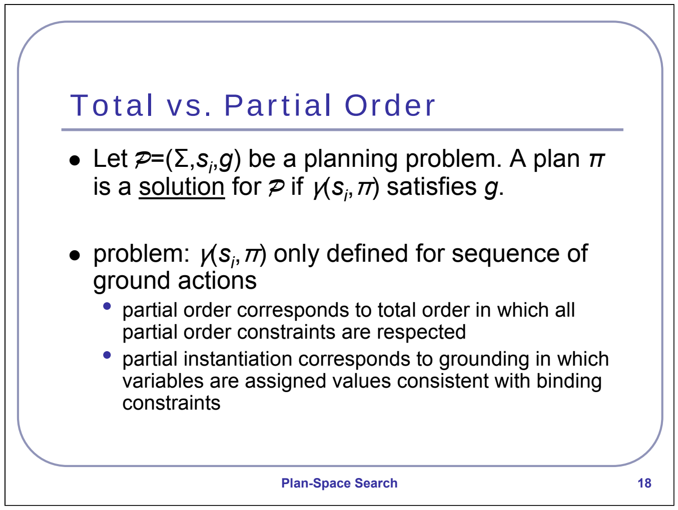
•successors must be consistent: constraints in a partial plan must be satisfiable

•plan-space planning decouple two sub-problems:

- •which actions need to be performed
- •how to organize these actions

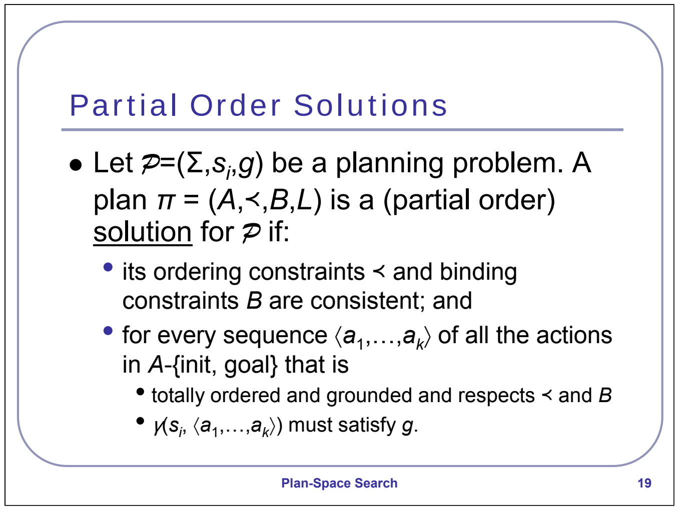
•partial plan as set of plans: refinement operation reduces the set to smaller subset

•next: to define planning as plan-space search problem: need to define goal state



Total vs. Partial Order

- **Let $\mathcal{P}=(\Sigma,s_i,g)$ be a planning problem. A plan π is a <u>solution</u> for $\mathcal{P}$ if $\gamma(s_i,\pi)$ satisfies g.**
    - solution defined for state transition system
- **problem: $\gamma(s_i,\pi)$ only defined for sequence of ground actions**
    - **partial order corresponds to total order in which all partial order constraints are respected**
        - partial ordering is consistent iff it is free of loops
        - note: there may be an exponential number of total ordering consistent with a given partial ordering
    - **partial instantiation corresponds to grounding in which variables are assigned values consistent with binding constraints**
        - note: exponential combinatorics of assigning values to variables



Partial Order Solutions

•Let $\mathcal{P}=(\Sigma,s_i,g)$ be a planning problem. A plan π = $(A,\prec,B,L)$ is a (partial order) <u>solution</u> for $\mathcal{P}$ if:

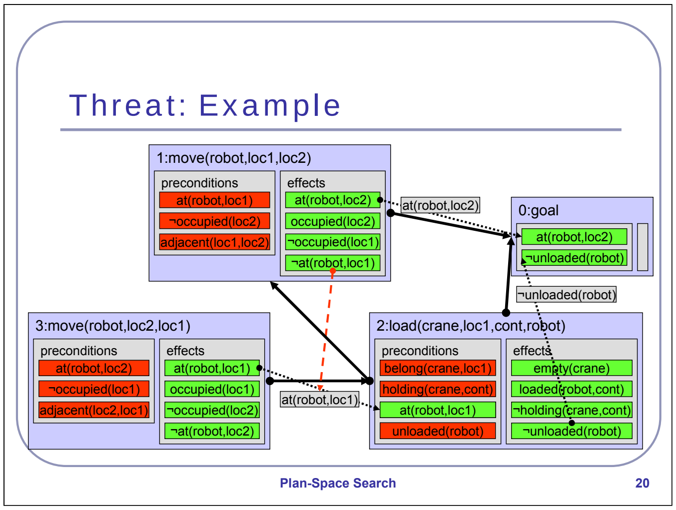
- **•its ordering constraints $\prec$ and binding constraints B are consistent; and**
- **•for every sequence $\langle a_1,\ldots,a_k\rangle$ of all the actions in A-{init, goal} that is**
  - **•totally ordered and grounded and respects $\prec$ and B**
  - **•$\gamma(s_i, \langle a_1,\ldots,a_k\rangle)$ must satisfy g.**

•note: causal links do not play a role in the definition of a solution

•with exponential number of sequences to check, definition is not very useful (as computational procedure for goal test)

•idea: use causal links to verify that every precondition of every action is supported by some other action

- •problem: condition not strong enough



Threat: Example

•start with partial plan from previous example (grounded; initial state not shown due to limited space on slide)

•introduce new 3:move action to achieve at(robot,loc1) precondition of 2:load action

  •note: still many unachieved preconditions – not a solution yet

•add causal link to maintain rationale

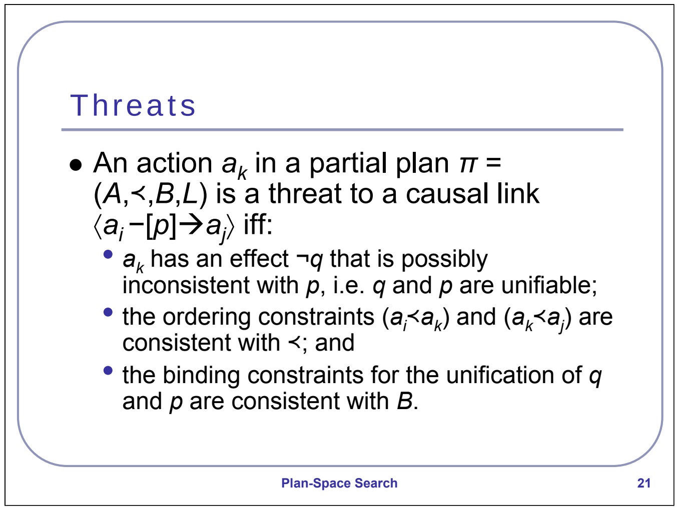
•add ordering to be consistent with causal link

•new: label causal link with condition it protects

•threat: effect of 1:move is negation of condition protected by causal link

  •if 1:move is executed between 3:move and 2:load the plan is no longer valid

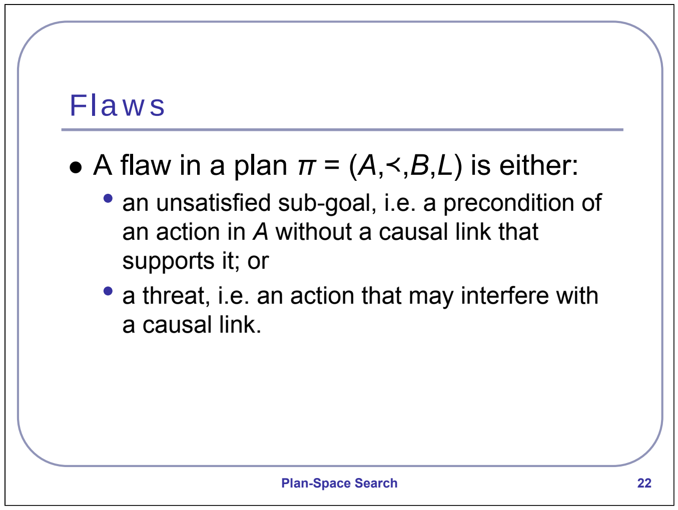
•possible solution: additional ordering constraint



Threats

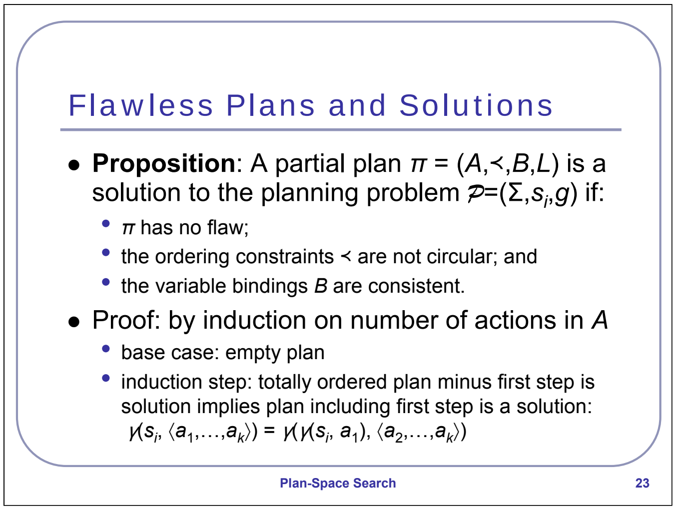
•An action a_k in a partial plan π = (A,$\prec$,B,L) is a threat to a causal link
$\langle a_i - [p] \rightarrow a_j \rangle$ iff:

- **•a_k has an effect $\neg q$ that is possibly inconsistent with p, i.e. q and p are unifiable;**
- **•the ordering constraints ($a_i \prec a_k$) and ($a_k \prec a_j$) are consistent with $\prec$; and**
- **•the binding constraints for the unification of q and p are consistent with B.**



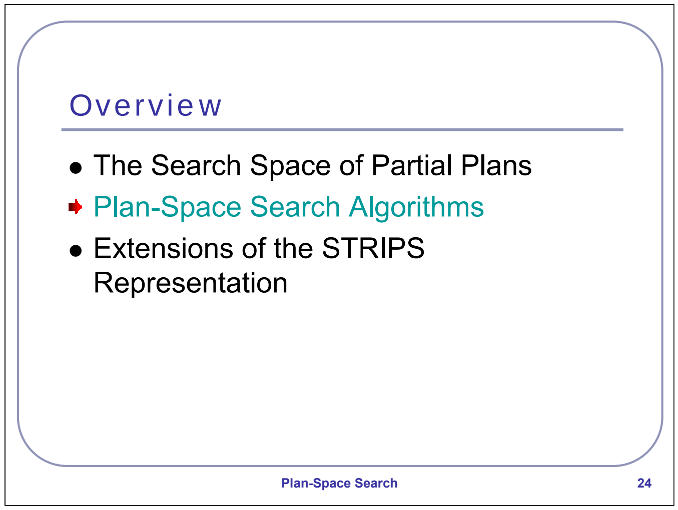
Flaws

- **A flaw in a plan $\pi = (A,\prec,B,L)$ is either:**
  - **an unsatisfied sub-goal, i.e. a precondition of an action in *A* without a causal link that supports it; or**
  - **a threat, i.e. an action that may interfere with a causal link.**



Flawless Plans and Solutions

- **Proposition: A partial plan π = (A,≺,B,L) is a solution to the planning problem $\mathcal{P}$=(Σ,s_i,g) if:**
  - **π has no flaw;**
  - **the ordering constraints ≺ are not circular; and**
  - **the variable bindings B are consistent.**
- computation:
  - let partial plans in the search space only violate the first condition (have flaws)
  - partial plans that violate either of the last two conditions cannot be refined into a solution and need not be generated
- **Proof: by induction on number of actions in A**
  - **base case: empty plan**
    - no flaws – every goal condition is supported by causal link from initial state
  - **induction step: totally ordered plan minus first step is solution implies plan including first step is a solution:** $\gamma(s_i, \langle a_1,\ldots,a_k \rangle) = \gamma(\gamma(s_i, a_1), \langle a_2,\ldots,a_k \rangle)$
    - truncated plan is solution to different problem



Overview

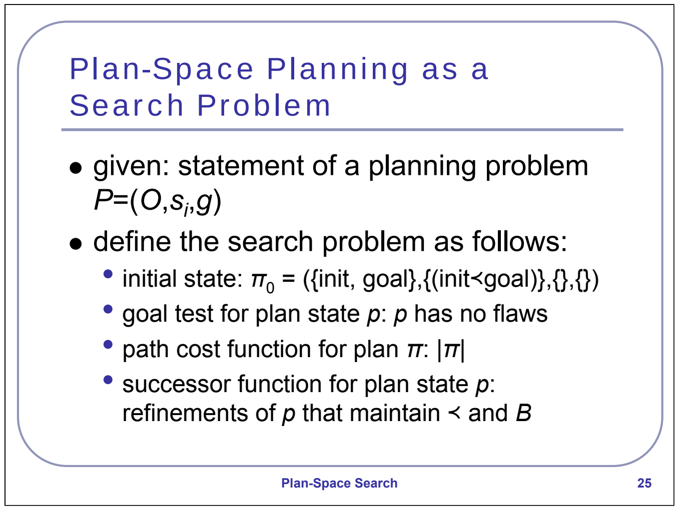
The Search Space of Partial Plans

- just done: defining the search space: partial plans + plan refinement operations

•Plan-Space Search Algorithms

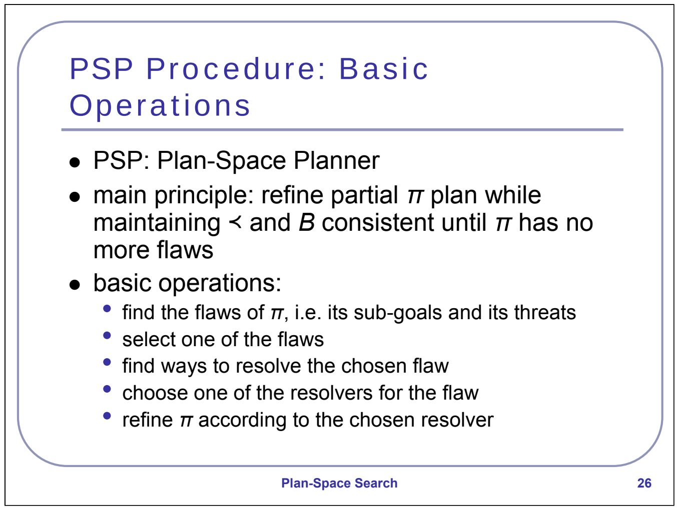
- •now: an algorithm that performs the search through the space of partial plans

•Extensions of the STRIPS Representation



Plan-Space Planning as a Search Problem

- **given: statement of a planning problem $P=(O,s_i,g)$**
- **define the search problem as follows:**
    - **initial state: π_0 = ({init, goal},{(init≺goal)},{},{})**
    - **goal test for plan state p: p has no flaws**
    - **path cost function for plan π: $|\pi|$**
    - **successor function for plan state p: refinements of p that maintain ≺ and B**
- note: plan space may be infinite even when state space is finite



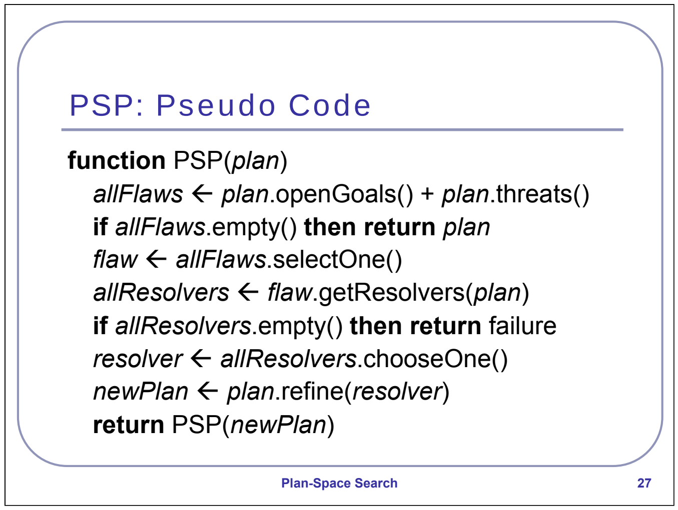
PSP Procedure: Basic Operations

•PSP: Plan-Space Planner

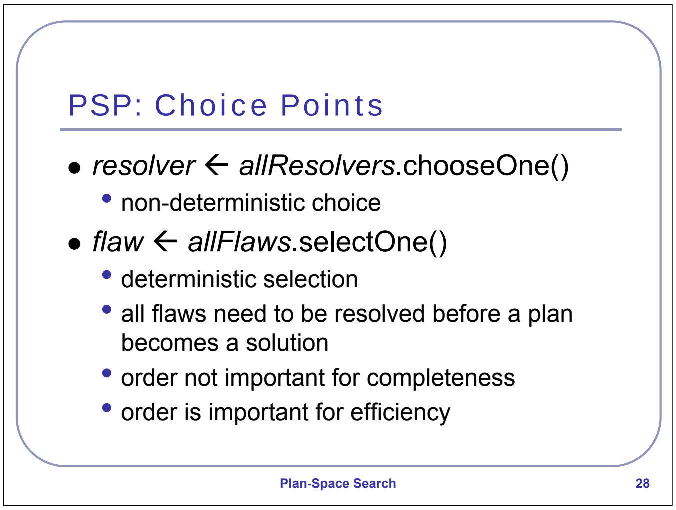
•main principle: refine partial *π* plan while maintaining $\prec$ and *B* consistent until *π* has no more flaws

•basic operations:

- **•find the flaws of *π*, i.e. its sub-goals and its threats**
  - •simple for empty plan – all goal conditions are unachieved sub-goals and no threats
- **•select one of the flaws**
- **•find ways to resolve the chosen flaw**
- **•choose one of the resolvers for the flaw**
- **•refine *π* according to the chosen resolver**
  - •modify the plan in such a way that $\prec$ and *B* are in a consistent state for the generated successor
    - •aim: no need to verify consistency of $\prec$ and *B* for goal test

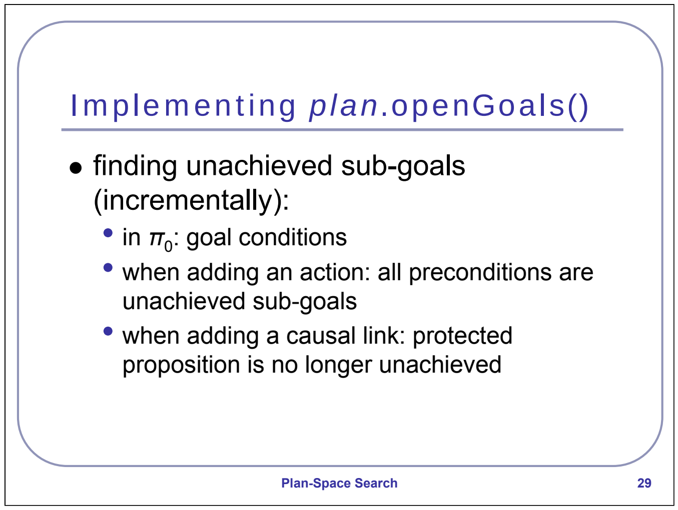


- **PSP: Pseudo Code**
- **function PSP(*plan*)**
    - refines the given partial plan into a solution plan; start with initial plan π_0
- ***allFlaws* ← *plan*.openGoals() + *plan*.threats()**
- **if *allFlaws*.empty() then return *plan***
    - see proposition in previous section: no flaws implies solution
- ***flaw* ← *allFlaws*.selectOne()**
- ***allResolvers* ← *flaw*.getResolvers(*plan*)**
    - represents all possible ways of removing the selected flaw from the partial plan
- **if *allResolvers*.empty() then return failure**
    - no resolvers means plan cannot be made flawless
- ***resolver* ← *allResolvers*.chooseOne()**
- ***newPlan* ← *plan*.refine(*resolver*)**
    - must maintain consistency of $\prec$ and *B*; new plan may contain new flaws
- **return PSP(*newPlan*)**



PSP: Choice Points

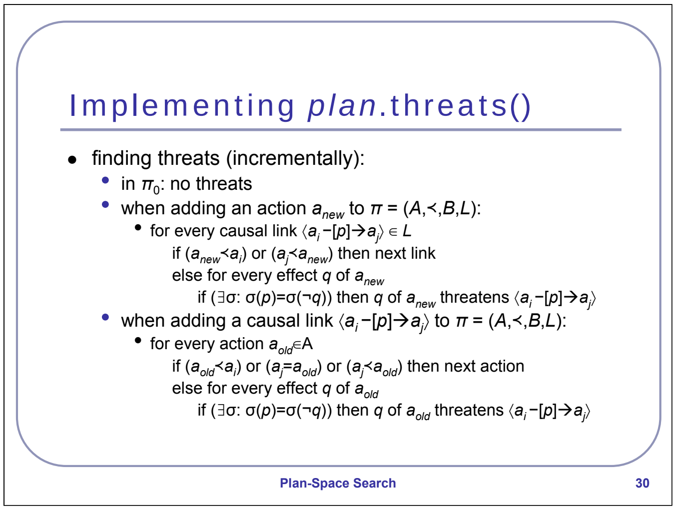
- ***resolver* ← *allResolvers*.chooseOne()**
  - **non-deterministic choice**
- ***flaw* ← *allFlaws*.selectOne()**
  - **deterministic selection**
  - **all flaws need to be resolved before a plan becomes a solution**
  - **order not important for completeness**
  - **order is important for efficiency**
    - for finding first plan, not so for finding all plans
    - deterministic implementation: using IDA*, for example



Implementing *plan*.openGoals()

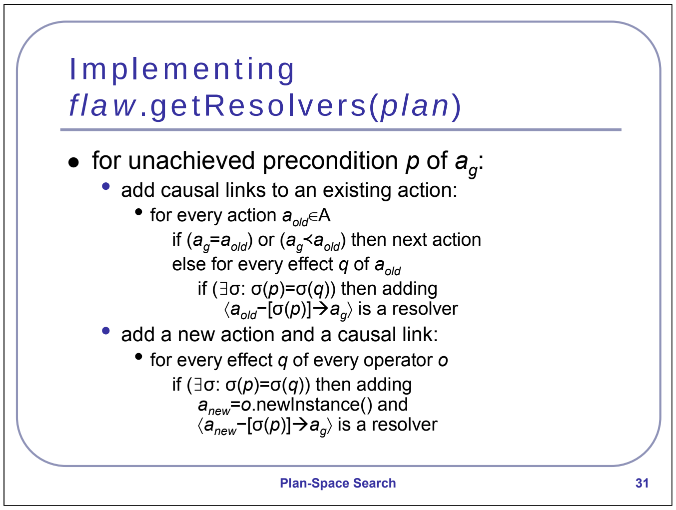
•finding unachieved sub-goals (incrementally):

- **in π_0: goal conditions**
- **when adding an action: all preconditions are unachieved sub-goals**
- **when adding a causal link: protected proposition is no longer unachieved**



Implementing *plan*.threats()

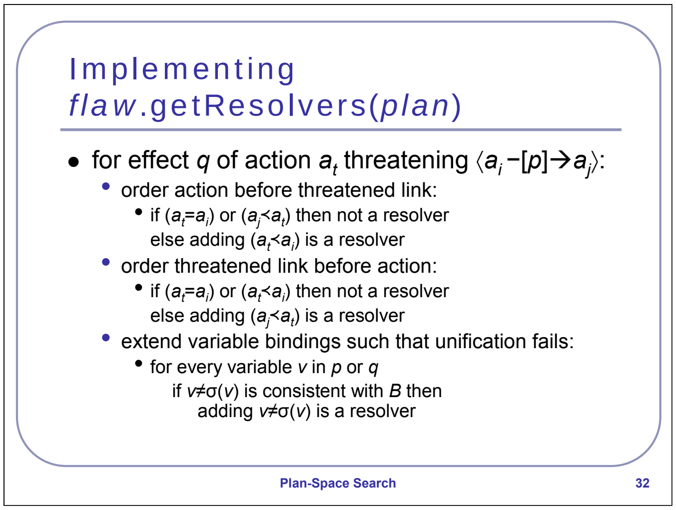
- **finding threats (incrementally):**
  - **in π_0: no threats**
  - **when adding an action a_{new} to $\pi = (A,\prec,B,L)$:**
    - **for every causal link $\langle a_i -[p]\rightarrow a_j \rangle \in L$**
    - **if $(a_{new} \prec a_i)$ or $(a_j \prec a_{new})$ then next link**
      - if the new action must occur before the provider or after the consumer of the link
    - **else for every effect q of a_{new}**
    - **if $(\exists\sigma: \sigma(p)=\sigma(\neg q))$ then q of a_{new} threatens $\langle a_i -[p]\rightarrow a_j \rangle$**
      - $\exists\sigma$: test whether there is a substitution consistent with *B*!
  - **when adding a causal link $\langle a_i -[p]\rightarrow a_j \rangle$ to $\pi = (A,\prec,B,L)$:**
    - **for every action $a_{old} \in A$**
    - **if $(a_{old} \prec a_i)$ or $(a_j = a_{old})$ or $(a_j \prec a_{old})$ then next action**
    - **else for every effect q of a_{old}**
    - **if $(\exists\sigma: \sigma(p)=\sigma(\neg q))$ then q of a_{old} threatens $\langle a_i -[p]\rightarrow a_j \rangle$**



Implementing *flaw*.getResolvers(*plan*)

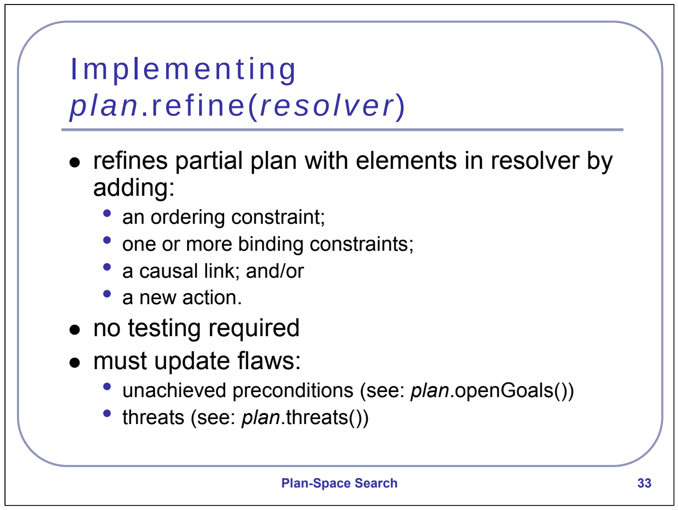
•for unachieved precondition p of a_g:

- **•add causal links to an existing action:**
    - **•for every action $a_{old} \in A$**
    - **•if ($a_g = a_{old}$) or ($a_g \prec a_{old}$) then next action**
    - **•else for every effect q of a_{old}**
    - **•if ($\exists\sigma$: $\sigma(p)=\sigma(q)$) then adding $\langle a_{old} - [\sigma(p)] \rightarrow a_g \rangle$ is a resolver**
- **•add a new action and a causal link:**
    - **•for every effect q of every operator o**
    - **•if ($\exists\sigma$: $\sigma(p)=\sigma(q)$) then adding a_{new}=o.newInstance() and $\langle a_{new} - [\sigma(p)] \rightarrow a_g \rangle$ is a resolver**



Implementing *flaw*.getResolvers(*plan*)

- **for effect q of action a_t threatening $\langle a_i -[p]\rightarrow a_j \rangle$:**
  - **order action before threatened link:**
    - **if (a_t=a_i) or ($a_j \prec a_t$) then not a resolver**
    - **else adding ($a_t \prec a_i$) is a resolver**
  - **order threatened link before action:**
    - **if (a_t=a_i) or ($a_t \prec a_i$) then not a resolver**
    - **else adding ($a_j \prec a_t$) is a resolver**
  - **extend variable bindings such that unification fails:**
    - **for every variable v in p or q**
    - **if $v \neq \sigma(v)$ is consistent with B then adding $v \neq \sigma(v)$ is a resolver**



Implementing *plan*.refine(*resolver*)

•refines partial plan with elements in resolver by adding:

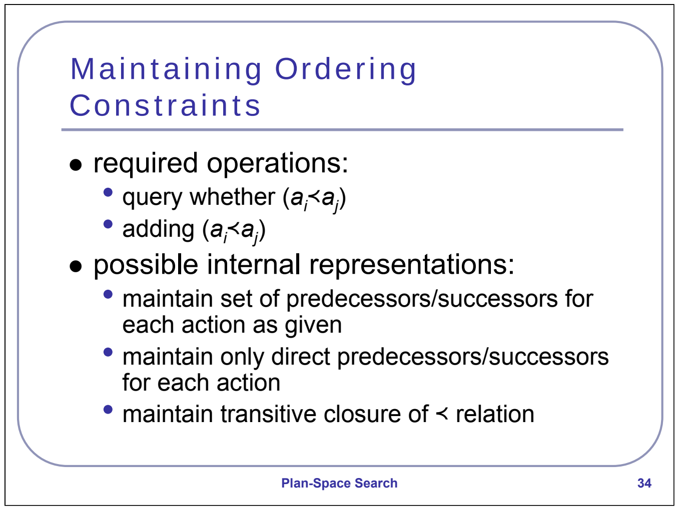
- **•an ordering constraint;**
- **•one or more binding constraints;**
- **•a causal link; and/or**
- **•a new action.**

•no testing required

- •all testing already done in *flaw*.getResolvers(*plan*)

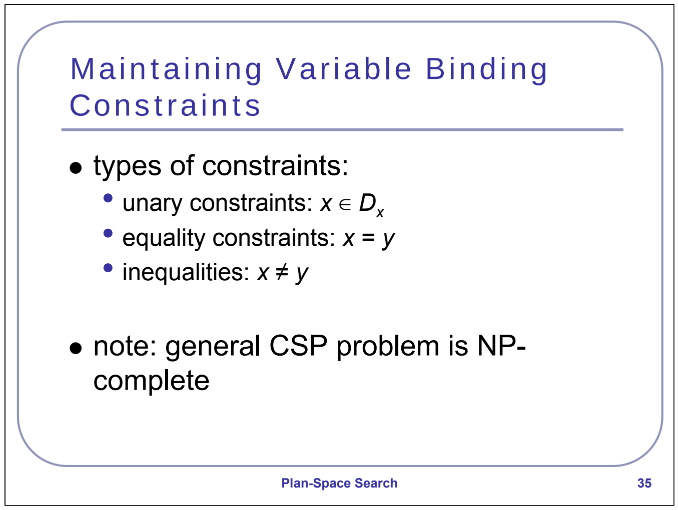
•must update flaws:

- **•unachieved preconditions (see: *plan*.openGoals())**
- **•threats (see: *plan*.threats())**



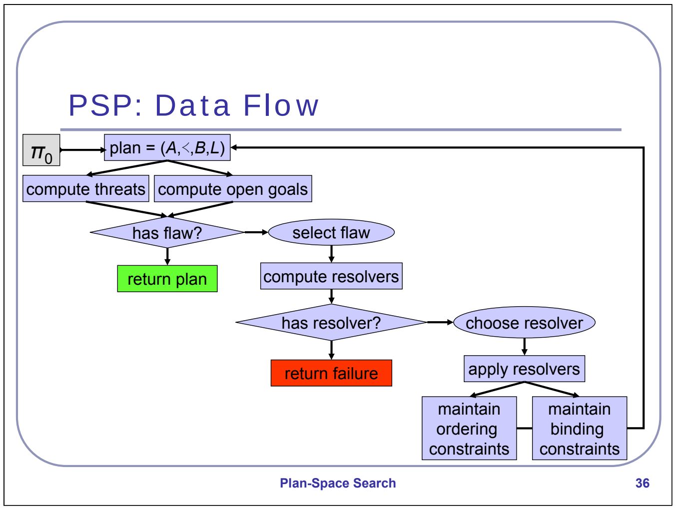
Maintaining Ordering Constraints

- **required operations:**
    - **query whether ($a_i \prec a_j$)**
    - **adding ($a_i \prec a_j$)**
        - without consistency testing
- **possible internal representations:**
    - **maintain set of predecessors/successors for each action as given**
    - **maintain only direct predecessors/successors for each action**
    - **maintain transitive closure of $\prec$ relation**
    - operations have different time and space complexity
- note: query performed more often than addition



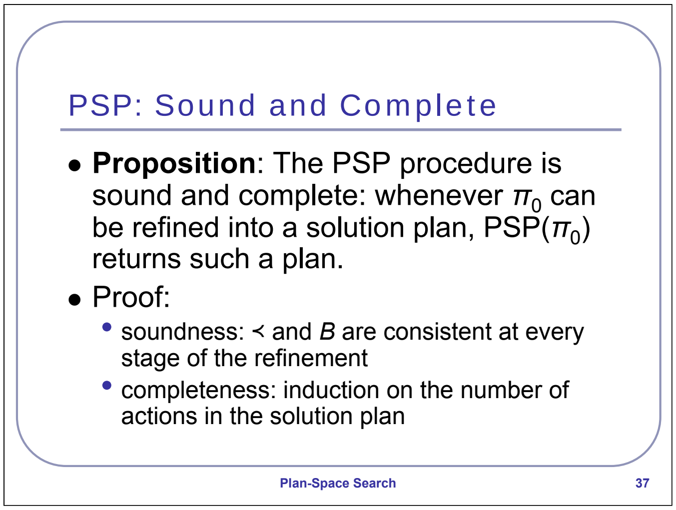
Maintaining Variable Binding Constraints

- **types of constraints:**
  - **unary constraints: $x \in D_x$**
  - **equality constraints: $x = y$**
    - unary and equality constraints can be solved in linear time
  - **inequalities: $x \neq y$**
    - inequalities give rise to general CSP problem
- **note: general CSP problem is NP-complete**



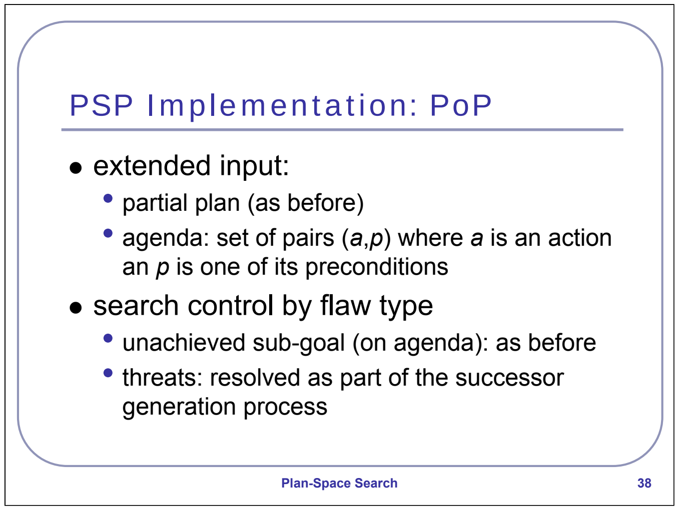
PSP: Data Flow

- •deterministic step: selecting a a flaw
    - •no backtracking required
    - •selection important for efficiency
    - •heuristic guidance required
- •non-deterministic step: choosing a resolver for a flaw
    - •implemented as backtracking
        - •order in which resolvers are tried important for efficiency
        - •heuristic guidance required
- •note: admissible heuristics (A*) must have step cost greater than zero



PSP: Sound and Complete

- **Proposition: The PSP procedure is sound and complete: whenever π_0 can be refined into a solution plan, PSP(π_0) returns such a plan.**
- **Proof:**
  - **soundness: ≺ and *B* are consistent at every stage of the refinement**
  - **completeness: induction on the number of actions in the solution plan**
    - note: non-deterministic version is complete, deterministic implementation must avoid infinite branches



PSP Implementation: PoP

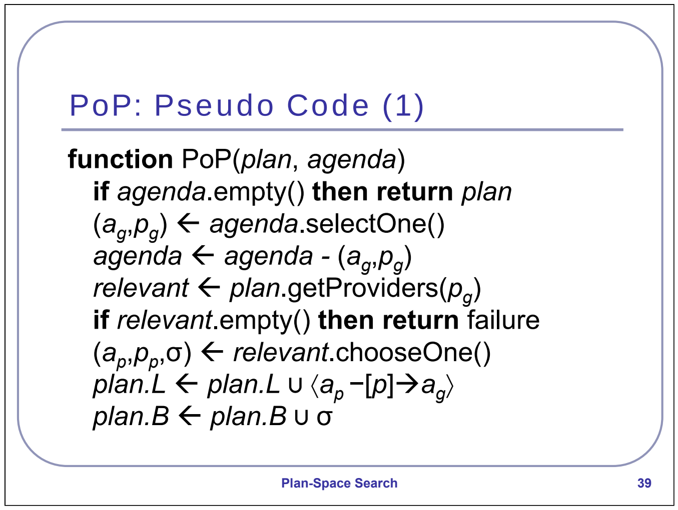
•based on UCPOP

•extended input:

- **•partial plan (as before)**
- **•agenda: set of pairs (*a*,*p*) where *a* is an action an *p* is one of its preconditions**
- •initial agenda: one pair for each precondition of the goal step

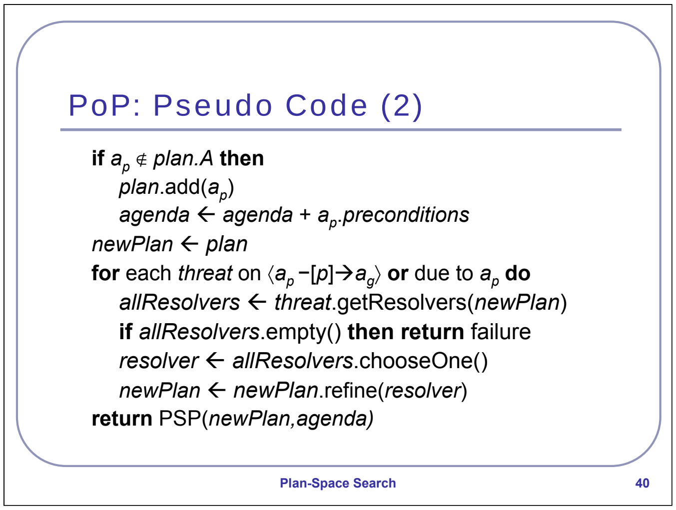
•search control by flaw type

- **•unachieved sub-goal (on agenda): as before**
- **•threats: resolved as part of the successor generation process**



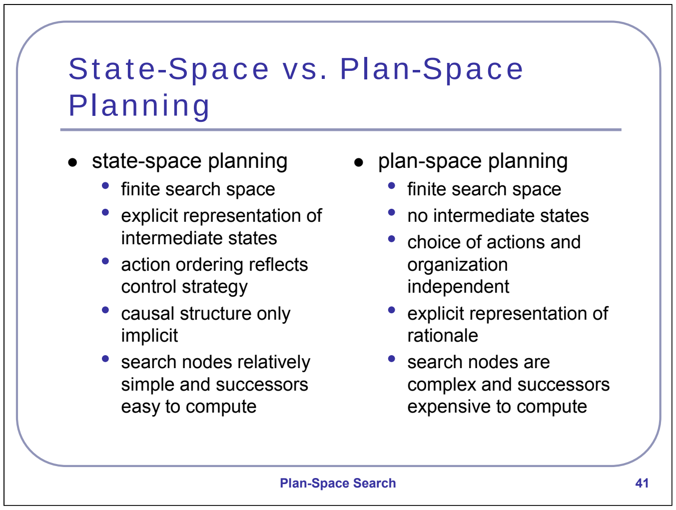
PoP: Pseudo Code (1)

- **function PoP(*plan*, *agenda*)**
- **if *agenda*.empty() then return *plan***
- **(a_g,p_g) ← *agenda*.selectOne()**
  - deterministic choice point
- ***agenda* ← *agenda* - (a_g,p_g)**
- ***relevant* ← *plan*.getProviders(p_g)**
  - finds all actions
    - either from within the *plan* or
    - from new instances of an operator
  - that have an effect that unifies with *condition*
- **if *relevant*.empty() then return failure**
- **(a_p,p_p,σ) ← *relevant*.chooseOne()**
  - non-deterministic choice point
- ***plan.L* ← *plan.L* ∪ ⟨a_p −[*p*]→ a_g⟩**
- ***plan.B* ← *plan.B* ∪ σ**
  - must succeed for elements of relevant



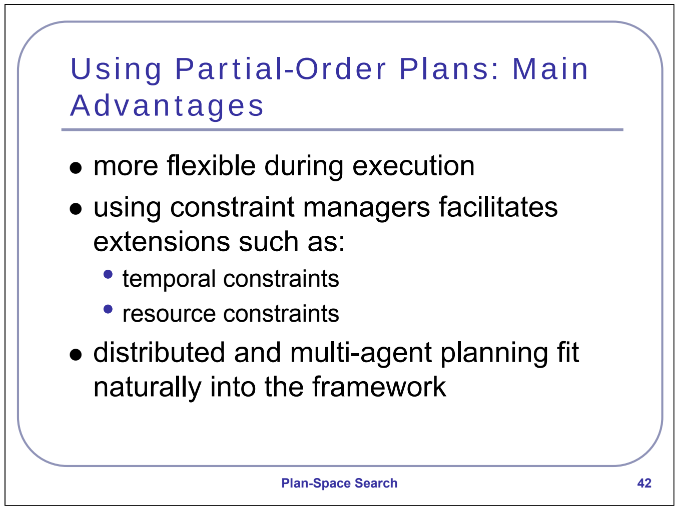
PoP: Pseudo Code (2)

- **if $a_p \notin$ *plan.A* then**
  - if the action is new and needs to be added to the plan
- ***plan*.add(a_p)**
  - involves updating set of actions and ordering constraints
- ***agenda* ← *agenda* + a_p.*preconditions***
  - all preconditions of the new action are new sub-goals
- ***newPlan* ← *plan***
- **for each *threat* on $\langle a_p -[p]\rightarrow a_g \rangle$ or due to a_p do**
  - note: two sources of threats are treated identically
- ***allResolvers* ← *threat*.getResolvers(*newPlan*)**
- **if *allResolvers*.empty() then return failure**
- ***resolver* ← *allResolvers*.chooseOne()**
  - second non-deterministic choice point
- ***newPlan* ← *newPlan*.refine(*resolver*)**
  - note: loop does not add to agenda
- **return PSP(*newPlan*,*agenda*)**



State-Space vs. Plan-Space Planning

- **state-space planning vs. plan-space planning**
  - **finite search space vs. finite search space**
    - important: portion of search space explored/generated
  - **explicit representation of intermediate states vs. no intermediate states**
    - explicit representation allows for efficient domain specific heuristics and control knowledge
  - **action ordering reflects control strategy vs. choice of actions and organization independent**
  - **causal structure only implicit vs. explicit representation of rationale**
    - important for plan execution
  - **search nodes relatively simple and successors easy to compute vs. search nodes are complex and successors expensive to compute**



Using Partial-Order Plans: Main Advantages

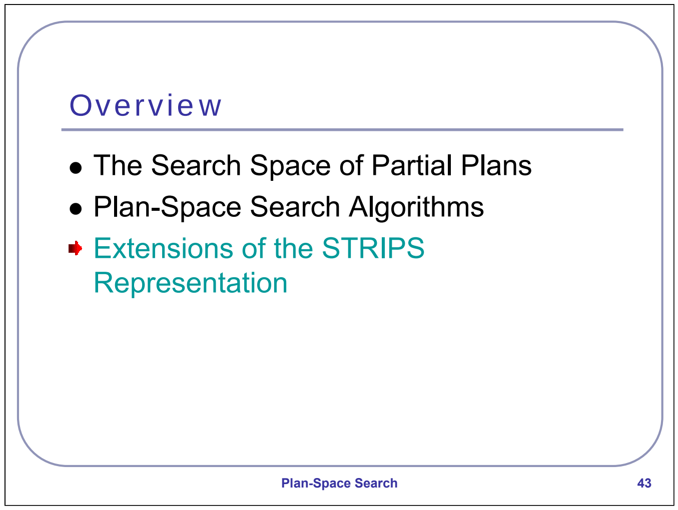
•more flexible during execution

- •saved rationale facilitates execution monitoring and re-planning

•using constraint managers facilitates extensions such as:

- **•temporal constraints**
- **•resource constraints**
  - •both very important for realistic planning

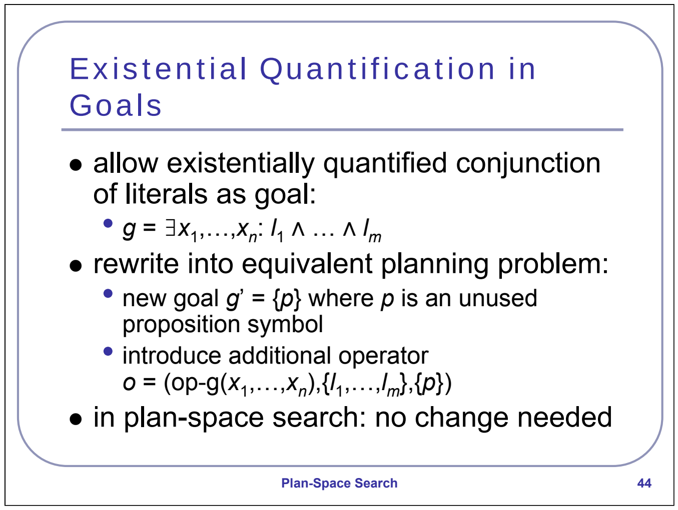
•distributed and multi-agent planning fit naturally into the framework



Overview

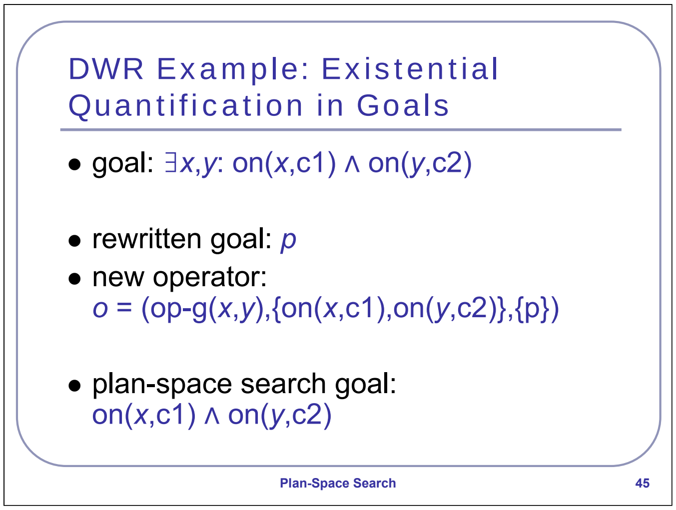
The Search Space of Partial Plans

- **Plan-Space Search Algorithms**
  - just done: an algorithm that performs the search through the space of partial plans
- **Extensions of the STRIPS Representation**
  - now: extensions to the restricted STRIPS representation and approaches to deal with them



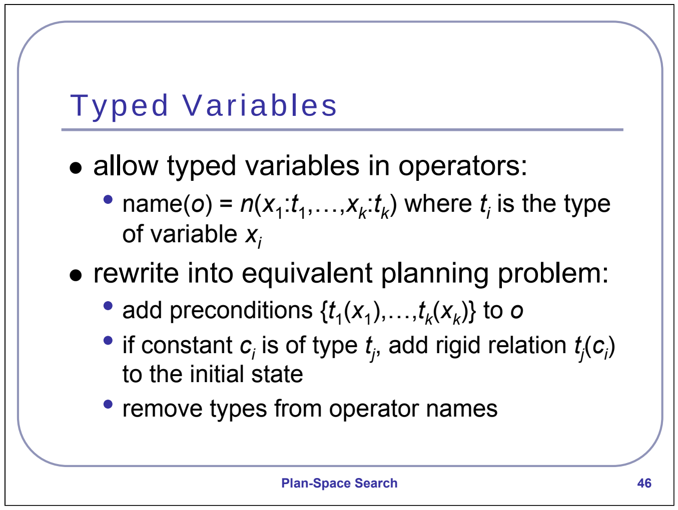
Existential Quantification in Goals

- **allow existentially quantified conjunction of literals as goal:**
  - $g = \exists\ x_1,\ldots,x_n : l_1 \wedge \ldots \wedge l_m$
    - but still no free variables in the goal literals
- **rewrite into equivalent planning problem:**
  - **new goal $g' = \{p\}$ where p is an unused proposition symbol**
  - **introduce additional operator $o = (\text{op-g}(x_1,\ldots,x_n),\{l_1,\ldots,l_m\},\{p\})$**
    - solution plans will be one step longer, with last step being an instantiation of op-g
- no increase in expressive power (can rewrite every problem), but sometimes makes representation appear more natural
- **in plan-space search: no change needed**
  - partial plans do not require all variables to be instantiated



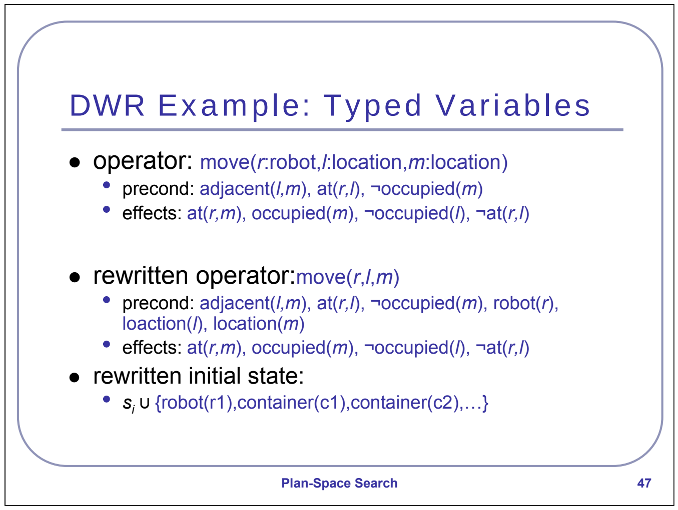
Example: Existential Quantification in Goals

- **goal: ∃*x*,*y*: on(*x*,c1) ∧ on(*y*,c2)**
- **rewritten goal: *p***
- **new operator: *o* = (op-g(*x*,*y*),{on(*x*,c1),on(*y*,c2)},{p})**
- **plan-space search goal: on(*x*,c1) ∧ on(*y*,c2)**
    - variables in goals are implicitly existentially quantified



Typed Variables

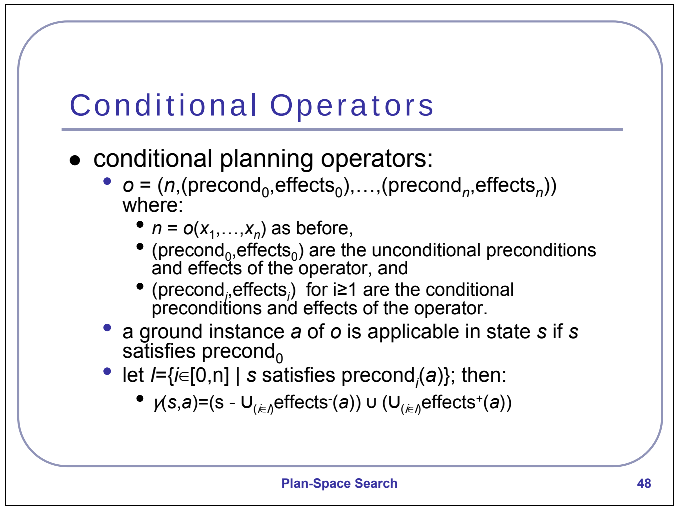
- **allow typed variables in operators:**
  - **name(o) = $n(x_1{:}t_1,\ldots,x_k{:}t_k)$ where t_i is the type of variable x_i**
    - types usually given with problem specification
    - exact syntax for typing not important
- **rewrite into equivalent planning problem:**
  - **add preconditions $\{t_1(x_1),\ldots,t_k(x_k)\}$ to o**
    - types are unary predicates here
  - **if constant c_i is of type t_j, add $t_j(c_i)$ to the initial state**
  - **remove types from operator names**
- similarly: typed relations
- advantages: readability, reduced number of actions (operator instances)



DWR Example: Typed Variables

- **operator: move(*r*:robot,*l*:location,*m*:location)**
  - **precond: adjacent(*l,m*), at(*r,l*), ¬occupied(*m*)**
  - **effects: at(*r,m*), occupied(*m*), ¬occupied(*l*), ¬at(*r,l*)**
- **rewritten operator:move(*r*,*l*,*m*)**
  - **precond: adjacent(*l,m*), at(*r,l*), ¬occupied(*m*), robot(*r*), loaction(*l*), location(*m*)**
  - **effects: at(*r,m*), occupied(*m*), ¬occupied(*l*), ¬at(*r,l*)**
- **rewritten initial state:**
  - **s_i ∪ {robot(r1),container(c1),container(c2),…}**

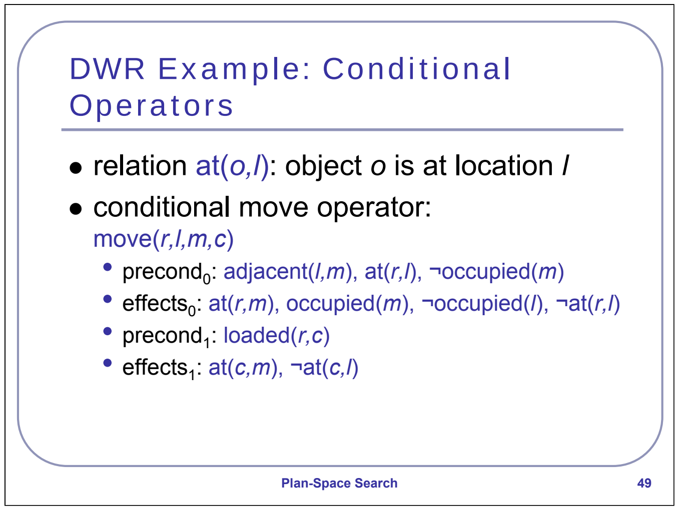
- note: dealing with typed variables directly in the planner is far more efficient



Conditional Operators

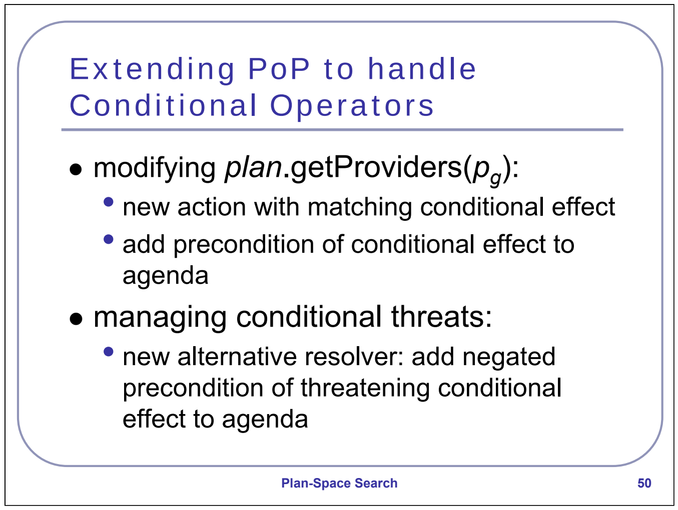
•conditional planning operators:

- **•*o* = (*n*,(precond$_0$,effects$_0$),…,(precond$_n$,effects$_n$)) where:**
  - **•$n = o(x_1,\dots,x_n)$ as before,**
  - **•(precond$_0$,effects$_0$) are the unconditional preconditions and effects of the operator, and**
    - •not strictly necessary, could be added to every conditional pair
  - **•(precond$_i$,effects$_i$) for i≥1 are the conditional preconditions and effects of the operator.**
- **•a ground instance *a* of *o* is applicable in state *s* if *s* satisfies precond$_0$**
- **•let I={i∈[0,n] | *s* satisfies precond$_i$(a)}; then:**
  - **•$\gamma(s,a)$=(s - $\cup_{(i\in I)}$effects$^-(a)$) ∪ ($\cup_{(i\in I)}$effects$^+(a)$)**



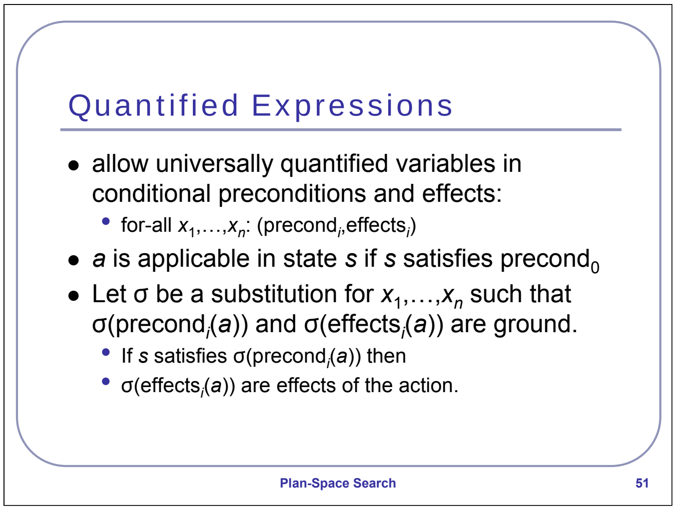
DWR Example: Conditional Operators

- **relation at(*o,l*): object *o* is at location *l***
    - at previously only used for location of robot
- **conditional move operator:**
move(*r,l,m,c*)
    - new parameter *c*, the potentially loaded container
    - **precond$_0$: adjacent(*l,m*), at(*r,l*), ¬occupied(*m*)**
    - **effects$_0$: at(*r,m*), occupied(*m*), ¬occupied(*l*), ¬at(*r,l*)**
        - as before
    - **precond$_1$: loaded(*r,c*)**
    - **effects$_1$: at(*c,m*), ¬at(*c,l*)**
        - if the container is loaded it will move with the robot



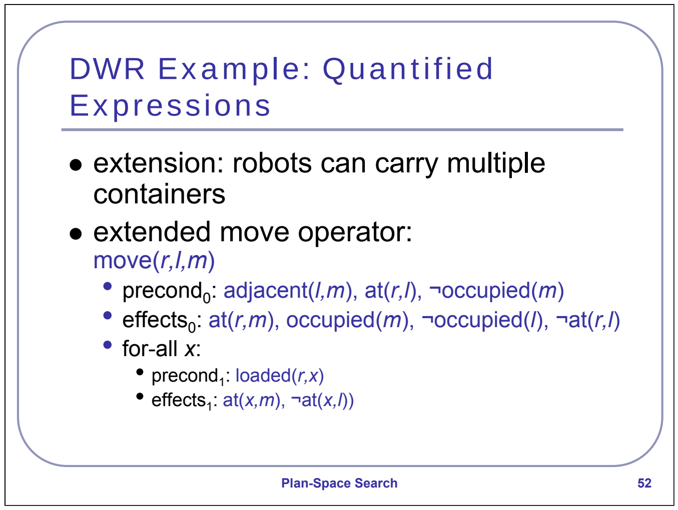
Extending PoP to handle Conditional Operators

- **modifying *plan*.getProviders(p_g):**
    - **new action with matching conditional effect**
    - **add precondition of conditional effect to agenda**
        - along with unconditional preconditions
- **managing conditional threats:**
    - when the threatening effect is a conditional effect
    - **new alternative resolver: add negated precondition of threatening conditional effect to agenda**
        - other alternatives: ordering and binding constraints



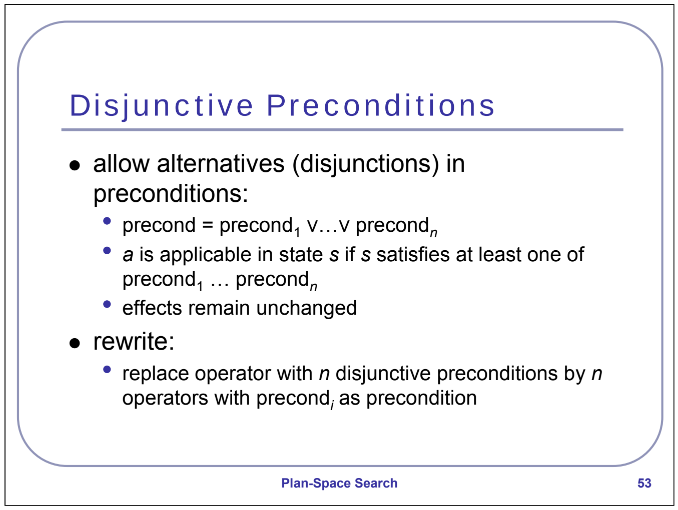
Quantified Expressions

- **allow universally quantified variables in conditional preconditions and effects:**
  - **for-all $x_1,\ldots,x_n$: (precond$_i$,effects$_i$)**
- ***a* is applicable in state *s* if *s* satisfies precond$_0$**
  - applicability depends only on unconditional preconditions
- **Let σ be a substitution for $x_1,\ldots,x_n$ such that σ(precond$_i$(a)) and σ(effects$_i$(a)) are ground.**
  - **If *s* satisfies σ(precond$_i$(a)) then**
  - **σ(effects$_i$(a)) are effects of the action.**
  - conditional effect occurs for all possible substitutions where the state satisfies the conditional preconditions
- note: cannot use binding constraints to resolve conditional threats here



DWR Example: Quantified Expressions

- **extension: robots can carry multiple containers**
- **extended move operator: move(*r*,*l*,*m*)**
  - container is no longer a parameter
  - **precond$_0$: adjacent(*l*,*m*), at(*r*,*l*), ¬occupied(*m*)**
  - **effects$_0$: at(*r*,*m*), occupied(*m*), ¬occupied(*l*), ¬at(*r*,*l*)**
  - **for-all *x*:** (the containers)
    - **precond$_1$: loaded(*r*,*x*)**
    - **effects$_1$: at(*x*,*m*), ¬at(*x*,*l*))**



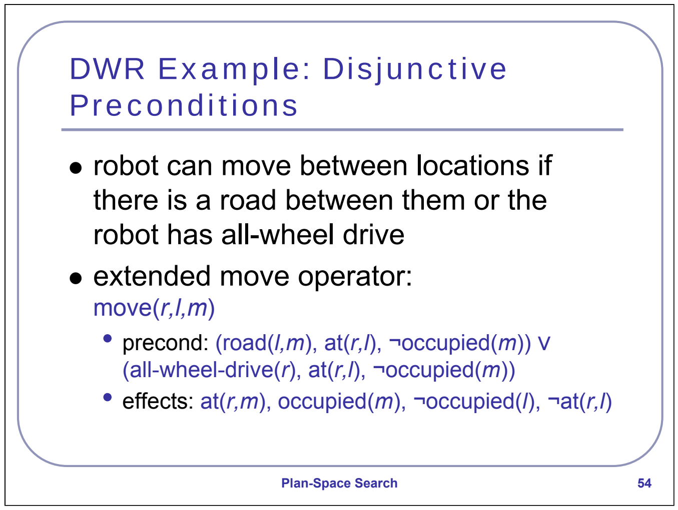
Disjunctive Preconditions

•allow alternatives (disjunctions) in preconditions:

- **precond = precond$_1$ ∨…∨ precond$_n$**
- ***a* is applicable in state *s* if *s* satisfies at least one of precond$_1$ … precond$_n$**
- **effects remain unchanged**

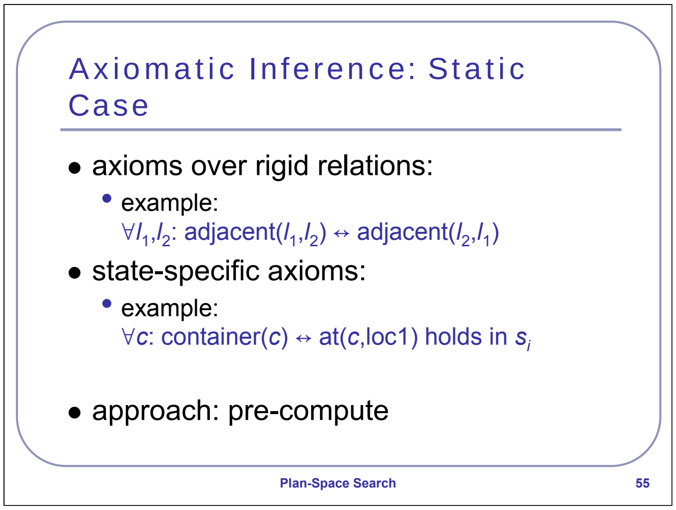
•rewrite:

- **replace operator with *n* disjunctive preconditions by *n* operators with precond$_i$ as precondition**
- leads to exponentially larger search space; more efficiently handled in the planner



DWR Example: Disjunctive Preconditions

- **robot can move between locations if there is a road between them or the robot has all-wheel drive**
- **extended move operator:**
  move(*r,l,m*)
  - **precond: (road(*l,m*), at(*r,l*), ¬occupied(*m*)) ∨ (all-wheel-drive(*r*), at(*r,l*), ¬occupied(*m*))**
    - more complex formulae can be transformed into DNF
  - **effects: at(*r,m*), occupied(*m*), ¬occupied(*l*), ¬at(*r,l*)**



Axiomatic Inference: Static Case

•idea: represent knowledge that is not explicit in the state of the world; derived knowledge

•**axioms over rigid relations:**

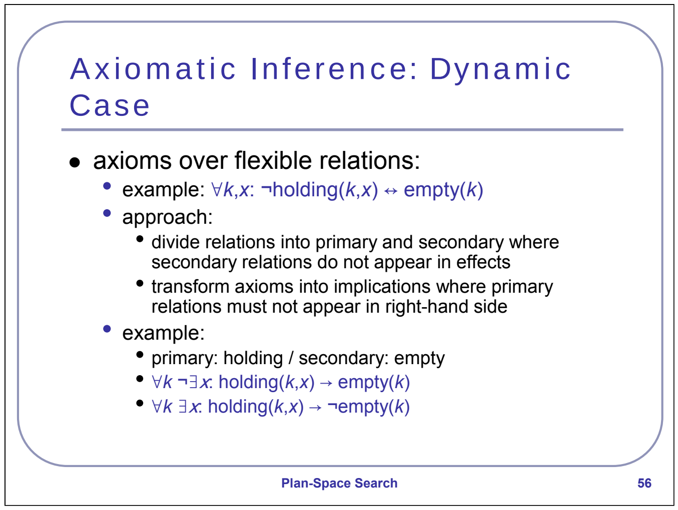
- •**example: $\forall l_1,l_2$: adjacent(l_1,l_2) ↔ adjacent(l_2,l_1)**
  - •adjacent is a symmetric relationship

•**state-specific axioms:**

- •**example: $\forall c$: container(c) ↔ at(c,loc1) holds in s_i**
  - •in the initial state, all containers are at location loc1

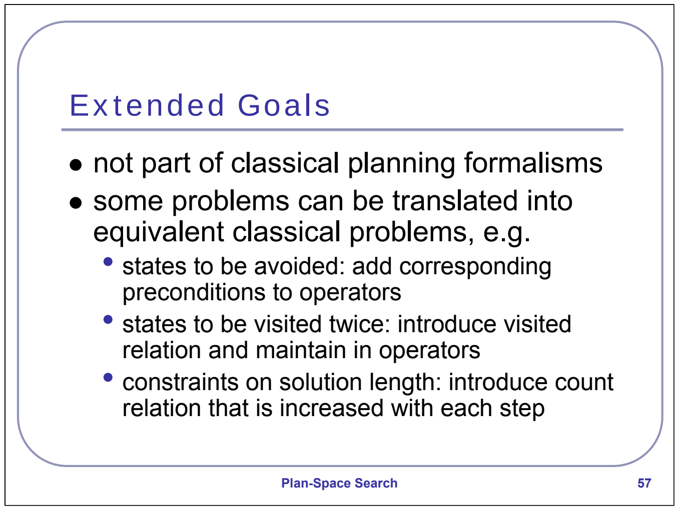
•**approach: pre-compute**

- •rigid relations: cannot appear in effects, truth value does not change from state to state; hence, can be pre-computed
- •state-specific axioms: expansion into ground atoms possible because of finite domains
  - •same technique for quantified effects



Axiomatic Inference: Dynamic Case

- **axioms over flexible relations:**
  - **example: $\forall k,x$: ¬holding(k,x) ↔ empty(k)**
    - a crane is empty iff it is not holding anything
  - **approach:**
    - **divide relations into primary and secondary where secondary relations do not appear in effects**
      - primary relations may appear in preconditions and effects, secondary relations only in preconditions
    - **transform axioms into implications where primary relations must not appear in right-hand side**
      - truth value of secondary relations depends on primary relations which are modified by operators
  - **example:**
    - **primary: holding / secondary: empty**
      - must remove empty-relation from effects of all operators
    - **$\forall k\ \neg\exists x$: holding(k,x) → empty(k)**
    - **$\forall k\ \exists x$: holding(k,x) → ¬empty(k)**

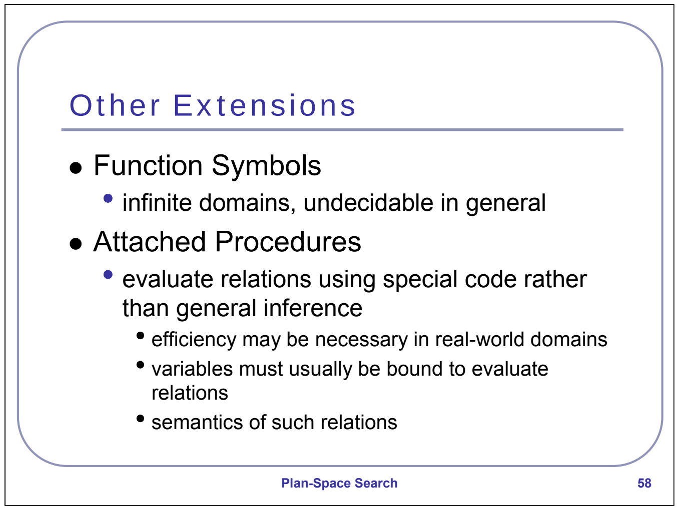


Extended Goals

•not part of classical planning formalisms

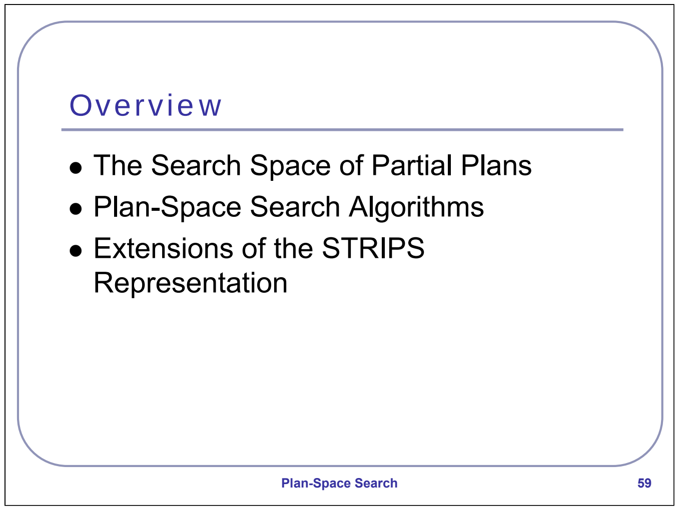
•some problems can be translated into equivalent classical problems, e.g.

- **•states to be avoided: add corresponding preconditions to operators**
- **•states to be visited twice: introduce visited relation and maintain in operators**
- **•constraints on solution length: introduce count relation that is increased with each step**
  - •requires function symbols and integer arithmetic (attached procedures)



Other Extensions

- **Function Symbols**
  - **infinite domains, undecidable in general**
    - expansion into ground atoms no longer possible, reasoning within one state is undecidable
- **Attached Procedures**
  - **evaluate relations using special code rather than general inference**
    - **efficiency may be necessary in real-world domains**
      - example: numeric computations
    - **variables must usually be bound to evaluate relations**
      - difficult with least commitment approach taken by many planners
    - **semantics of such relations**



Overview

➧**The Search Space of Partial Plans**

•**Plan-Space Search Algorithms**

•**Extensions of the STRIPS Representation**

- •just done: extensions to the restricted STRIPS representation and approaches to deal with them