Artificial Intelligence Planning

Plan-Space Search
Example: Partial Plan

1: move(robot, loc1, loc2)
   - preconditions: adjacent(loc1, loc2)
   - effects: at(robot, loc2), ¬occupied(loc2), at(robot, loc1), ¬occupied(loc1)

2: load(crane, loc1, cont, robot)
   - preconditions: belong(crane, loc1), holding(crane, cont), at(robot, loc1), empty(crane), loaded(robot, cont)
   - effects: empty(crane), ¬holding(crane, cont), ¬loaded(robot, cont)

3: move(robot, loc2, loc1)
   - preconditions: adjacent(loc2, loc1)
   - effects: at(robot, loc1), ¬occupied(loc1), at(robot, loc2), ¬occupied(loc2)

0: goal
   - at(robot, loc2), ¬unloaded(robot)
Overview

- Search States: Partial Plans
- Plan Refinement Operations
- The Plan-Space Search Problem
- Flawless Partial Plans
- The PSP Algorithm
- PSP Implementation Details
- Partial-Order Planning

Search States: Partial Plans

- now: introducing a completely different search space with partial plans as search states

Plan Refinement Operations

The Plan-Space Search Problem

Flawless Partial Plans

The PSP Algorithm

PSP Implementation Details

Partial-Order Planning
State-Space vs. Plan-Space Search

• state-space search: search through graph of nodes representing world states
• plan-space search: search through graph of partial plans
  – nodes: partially specified plans
  – arcs: plan refinement operations
  – solutions: partial-order plans

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
Partial Plans

• plan: set of actions organized into some structure
• partial plan:
  – subset of the actions
  – subset of the organizational structure
    • temporal ordering of actions
    • rationale: what the action achieves in the plan
  – subset of variable bindings

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
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 $(a_i \rightarrow [p] \leftarrow a_j)$ 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-goals 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
Overview

• Search States: Partial Plans
• Plan Refinement Operations
• The Plan-Space Search Problem
• Flawless Partial Plans
• The PSP Algorithm
• PSP Implementation Details
• Partial-Order Planning

Overview

• Search States: Partial Plans
  • just done: introducing a completely different search space with partial plans as search states

➢ Plan Refinement Operations
  • now: state transitions in the new search space – refining partial plans

• The Plan-Space Search Problem
• Flawless Partial Plans
• The PSP Algorithm
• PSP Implementation Details
• Partial-Order Planning
Adding Actions

• partial plan contains actions
  – initial state
  – goal conditions
  – set of operators with different variables

• reason for adding new actions
  – to achieve unsatisfied preconditions
  – to achieve unsatisfied goal conditions

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

• reasons for adding causal links
  – prevent interference with other actions

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
  - solution plan must contain actions
  - variable binding constraints keep track of possible values for variables and co-designation
- 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: $l_1 \neq \text{loc2}$
• 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
- reasons for adding ordering constraints
  - all actions after initial state
  - all actions before goal
  - causal link implies ordering constraint
  - to avoid possible interference

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
Overview

- Search States: Partial Plans
- Plan Refinement Operations
- The Plan-Space Search Problem
- Flawless Partial Plans
- The PSP Algorithm
- PSP Implementation Details
- Partial-Order Planning

Overview
- Search States: Partial Plans
- Plan Refinement Operations
  - just done: state transitions in the new search space – refining partial plans
- The Plan-Space Search Problem
  - now: definition of the plan-space search problem and solutions
- Flawless Partial Plans
- The PSP Algorithm
- PSP Implementation Details
- Partial-Order Planning
Plan-Space Search: Initial Search State

• represent initial state and goal as dummy actions
  – init: no preconditions, initial state as effects
  – goal: goal conditions as preconditions, no effects

• empty plan \( \pi_0 = \langle \{\text{init}, \text{goal}\}, \{(\text{init} \prec \text{goal})\}, \{\}, \{\} \rangle \):
  – 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

• 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 \( \pi_0 = \langle \{\text{init}, \text{goal}\}, \{(\text{init} \prec \text{goal})\}, \{\}, \{\} \rangle \):
  • 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):
  – adding an action to $A$
  – adding an ordering constraint to $\prec$
  – adding a binding constraint to $B$
  – adding a causal link to $L$

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
Let \( \mathcal{P} = (\Sigma, s_0, g) \) be a planning problem. A plan \( \pi \) is a solution for \( \mathcal{P} \) if \( \gamma(s_0, \pi) \) satisfies \( g \).

**Problem:** \( \gamma(s_0, \pi) \) only defined for sequence of ground actions
- partial order corresponds to total order in which all partial order constraints are respected
- partial instantiation corresponds to grounding in which variables are assigned values consistent with binding constraints

Total vs. Partial Order

- Let \( \mathcal{P} = (\Sigma, s_0, g) \) be a planning problem. A plan \( \pi \) is a solution for \( \mathcal{P} \) if \( \gamma(s_0, \pi) \) satisfies \( g \).

  - solution defined for state transition system

  - problem: \( \gamma(s_0, \pi) \) 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 $\pi = (A, <, B, L)$ is a (partial order) solution for $\mathcal{P}$ if:
  - its ordering constraints $<$ 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 $<$ 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
Overview

- Search States: Partial Plans
- Plan Refinement Operations
- The Plan-Space Search Problem
- Flawless Partial Plans
- The PSP Algorithm
- PSP Implementation Details
- Partial-Order Planning

Overview

- Search States: Partial Plans
- Plan Refinement Operations
- The Plan-Space Search Problem
  - just done: definition of the plan-space search problem and solutions (without goal test)
- Flawless Partial Plans
  - now: the goal test that completes the search problem
- The PSP Algorithm
- PSP Implementation Details
- Partial-Order Planning
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 $\pi = (A,\prec, B, L)$ is a threat to a causal link $\langle a, \lnot[p]\rightarrow a_j \rangle$ iff:
  – $a_k$ has an effect $\lnot q$ 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 $B$. 

Threats

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

*•* $a_k$ has an effect $\lnot q$ 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 $\pi = (A, <, B, L)$ 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 \( \pi = (A, \prec, B, L) \) is a solution to the planning problem \( \mathcal{P} = (\Sigma, s_i, g) \) if:
  - \( \pi \) has no flaw;
  - the ordering constraints \( \prec \) are not circular; and
  - the variable bindings \( B \) are consistent.

• **Proof:** by induction on number of actions in \( A \)
  - base case: empty plan
  - induction step: totally ordered plan minus first step is solution implies plan including first step is a solution:
    \[
    \gamma(s_p, \langle a_1, \ldots, a_k \rangle) = \gamma(\gamma(s_p, a_1), \langle a_2, \ldots, a_k \rangle)
    \]

**Flawless Plans and Solutions**

• Proposition: A partial plan \( \pi = (A, \prec, B, L) \) is a solution to the planning problem \( \mathcal{P} = (\Sigma, s_i, g) \) if:
  • \( \pi \) has no flaw;
  • the ordering constraints \( \prec \) 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:
    \[
    \gamma(s_p, \langle a_1, \ldots, a_k \rangle) = \gamma(\gamma(s_p, a_1), \langle a_2, \ldots, a_k \rangle)
    \]
    • truncated plan is solution to different problem
Overview

• Search States: Partial Plans
• Plan Refinement Operations
• The Plan-Space Search Problem
• Flawless Partial Plans
• The PSP Algorithm
• PSP Implementation Details
• Partial-Order Planning

Overview
• Search States: Partial Plans
• Plan Refinement Operations
• The Plan-Space Search Problem
• Flawless Partial Plans
  • just done: the goal test that completes the search problem

➢ The PSP Algorithm
  • now: a generic plan-space search planning algorithm

• PSP Implementation Details
• Partial-Order Planning
Plan-Space Planning as a Search Problem

• given: statement of a planning problem $P = (O, s, g)$
• define the search problem as follows:
  – initial state: $\pi_0 = \{(\text{init}, \text{goal}), \{(\text{init} \prec \text{goal})\}, \emptyset, \emptyset\}$
  – goal test for plan state $\rho$: $\rho$ has no flaws
  – path cost function for plan $\pi$: $|\pi|$  
  – successor function for plan state $\rho$: refinements of $\rho$ that maintain $\prec$ 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 $\pi$ plan while maintaining $\prec$ and $B$ consistent until $\pi$ has no more flaws
• basic operations:
  – find the flaws of $\pi$, i.e. its sub-goals and its threats
  – select one of the flaws
  – find ways to resolve the chosen flaw
  – choose one of the resolvers for the flaw
  – refine $\pi$ according to the chosen resolver

PSP Procedure: Basic Operations
• PSP: Plan-Space Planner
• main principle: refine partial $\pi$ plan while maintaining $\prec$ and $B$ consistent until $\pi$ has no more flaws
• basic operations:
  • find the flaws of $\pi$, 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 $\pi$ 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)
    allFlaws ← plan.openGoals() + plan.threats()
    if allFlaws.empty() then return plan
    flaw ← allFlaws.selectOne()
    allResolvers ← flaw.getResolvers(plan)
    if allResolvers.empty() then return failure
    resolver ← allResolvers.chooseOne()
    newPlan ← plan.refine(resolver)
    return PSP(newPlan)

• PSP: Pseudo Code
• function PSP(plan)
  • refines the given partial plan into a solution plan; start with initial plan \( \pi_0 \)
• 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
Overview

• Search States: Partial Plans
• Plan Refinement Operations
• The Plan-Space Search Problem
• Flawless Partial Plans
• The PSP Algorithm
• PSP Implementation Details
• Partial-Order Planning

PSP Implementation Details

• just done: a generic plan-space search planning algorithm (overview)

• Partial-Order Planning
Implementing `plan.openGoals()`

• finding unachieved sub-goals (incrementally):
  – in $\pi_0$: goal conditions
  – when adding an action: all preconditions are unachieved sub-goals
  – when adding a causal link: protected proposition is no longer unachieved
Implementing $\text{plan.threats()}$

- finding threats (incrementally):
  - in $\pi_0$: no threats
  - when adding an action $a_{\text{new}}$ to $\pi = (A,\prec,B,L)$:
    - for every causal link $\langle a_i,[-p]\rightarrow a_j \rangle \in L$
      - if $(a_{\text{new}}\prec a_i)$ or $(a_j\prec a_{\text{new}})$ then next link
        - if $\exists \sigma: \sigma(p)=\sigma(\neg q)$ then $q$ of $a_{\text{new}}$ threatens $\langle a_i,[-p]\rightarrow a_j \rangle$
    - when adding a causal link $\langle a_i,[-p]\rightarrow a_j \rangle$ to $\pi = (A,\prec,B,L)$:
      - for every action $a_{\text{old}} \in A$
        - if $(a_{\text{old}}\prec a_i)$ or $(a_j=a_{\text{old}})$ or $(a_j\prec a_{\text{old}})$ then next action
          - else for every effect $q$ of $a_{\text{old}}$
            - if $\exists \sigma: \sigma(p)=\sigma(\neg q)$ then $q$ of $a_{\text{old}}$ 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 \( (a_g < a_{old}) \) then next action
      - else for every effect \( q \) of \( a_{old} \)
        - if \( (\exists \sigma: \sigma(p) = \sigma(q)) \) then adding
          \(<a_{old} - \sigma(p) \rightarrow a_g> \) 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
          \(<a_{new} - \sigma(p) \rightarrow a_g> \) is a resolver
Implementing `flaw.getResolvers(plan)`

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

Implementing `flaw.getResolvers(plan)`

- for effect \( q \) of action \( a_i \) threatening \( \langle a_i, [p] \rightarrow a_j \rangle \):
  - order action before threatened link:
    - if \( (a_i = a_j) \) or \( (a_j \prec a_i) \) then not a resolver
      else adding \( (a_i \prec a_j) \) is a resolver
  - order threatened link before action:
    - if \( (a_i = a_j) \) or \( (a_i \prec a_j) \) then not a resolver
      else adding \( (a_j \prec a_i) \) is a resolver
  - extend variable bindings such that unification fails:
    - for every variable \( v \) in \( p \) or \( q \)
      if \( v \not\equiv \sigma(v) \) is consistent with \( B \) then
        adding \( v \not\equiv \sigma(v) \) is a resolver
Implementing \textit{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
- must update flaws:
  - unachieved preconditions (see: \textit{plan.openGoals()})
  - threats (see: \textit{plan.threats()})
Maintaining Ordering Constraints

• required operations:
  – query whether \((a_i, a_j)\)
  – adding \((a_i, a_j)\)
• 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 \(<\) relation

Maintaining Ordering Constraints

• required operations:
  • query whether \((a_i, a_j)\)
  • adding \((a_i, a_j)\)
    • 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 \(<\) 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: \( x = y \)
  – inequalities: \( x \neq y \)

• note: general CSP problem is NP-complete

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: Sound and Complete

• **Proposition**: The PSP procedure is sound and complete: whenever \( \pi_0 \) can be refined into a solution plan, \( \text{PSP}(\pi_0) \) returns such a plan.

• **Proof**:  
  – soundness: \( \prec \) 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
Overview

• Search States: Partial Plans
• Plan Refinement Operations
• The Plan-Space Search Problem
• Flawless Partial Plans
• The PSP Algorithm
• PSP Implementation Details
• Partial-Order Planning

Overview

• Search States: Partial Plans
• Plan Refinement Operations
• The Plan-Space Search Problem
• Flawless Partial Plans
• The PSP Algorithm
• PSP Implementation Details
  • just done: functions for identifying flaws and resolving them (used in PSP)

Partial-Order Planning
  • now: the algorithm implemented by the UCPOP planner
PSP: Data Flow

• deterministic step: selecting 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 Implementation: PoP

- **extended input:**
  - partial plan (as before)
  - agenda: set of pairs \((a,p)\) where \(a\) is an action and \(p\) is one of its preconditions
- **search control by flaw type**
  - unachieved sub-goal (on agenda): as before
  - threats: resolved as part of the successor generation process

---

PSP Implementation: PoP

- based on UCPOP
- **extended input:**
  - partial plan (as before)
  - agenda: set of pairs \((a,p)\) where \(a\) is an action and \(p\) 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)**

```plaintext
function PoP(plan, agenda)
    if agenda.empty() then return plan
    (a_g, p_g) ← agenda.selectOne()
    agenda ← agenda - (a_g, p_g)
    relevant ← plan.getProviders(p_g)
    if relevant.empty() then return failure
    (a_p, p_p, σ) ← relevant.chooseOne()
    plan.L ← plan.L ∪ ⟨a_p - [σ(p_p)] → a_g⟩
    plan.B ← plan.B ∪ σ
```

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 $a_p \notin \text{plan.A}$ then
  \text{plan.add}(a_p)
  \text{agenda} \leftarrow \text{agenda} + a_p.\text{preconditions}
  \text{newPlan} \leftarrow \text{plan}
  \text{for each threat on } (a_p \rightarrow p) \text{ or due to } a_p \text{ do}
    \text{allResolvers} \leftarrow \text{threat.getResolvers(\text{newPlan})}
    \text{if allResolvers.empty() then return failure}
    \text{resolver} \leftarrow \text{allResolvers.chooseOne()}
    \text{newPlan} \leftarrow \text{newPlan.refine(resolver)}
  \text{return PoP(\text{newPlan,agenda})}

PoP: Pseudo Code (2)
• if $a_p \notin \text{plan.A}$ 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 $\leftarrow$ agenda + $a_p$.preconditions
      • all preconditions of the new action are new sub-goals
    • newPlan $\leftarrow$ plan
    • for each threat on $(a_p \rightarrow p)$ or due to $a_p$ do
      • note: two sources of threats are treated identically
    • allResolvers $\leftarrow$ threat.getResolvers(newPlan)
    • if allResolvers.empty() then return failure
    • resolver $\leftarrow$ allResolvers.chooseOne()
      • second non-deterministic choice point
    • newPlan $\leftarrow$ newPlan.refine(resolver)
      • note: loop does not add to agenda
    • return PSP(newPlan,agenda)
## State-Space vs. Plan-Space Planning

- **State-Space Planning**
  - Finite search space
  - Explicit representation of intermediate states
  - Action ordering reflects control strategy
  - Causal structure only implicit
  - Search nodes relatively simple and successors easy to compute

- **Plan-Space Planning**
  - Infinite search space
  - No intermediate states
  - Choice of actions and organization independent
  - Explicit representation of rationale
  - Search nodes are complex and successors expensive to compute

---

### State-Space vs. Plan-Space Planning

- **State-space planning vs. plan-space planning**
  - **Finite search space vs. infinite search space**
    - Important: portion of search space explored/generated; both search trees potentially infinite
  - **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**
Overview

• Search States: Partial Plans
• Plan Refinement Operations
• The Plan-Space Search Problem
• Flawless Partial Plans
• The PSP Algorithm
• PSP Implementation Details
• Partial-Order Planning

just done: the algorithm implemented by the UCPOP planner