The Graphplan Planner

Searching the Planning Graph

Literature

Neoclassical Planning

- concerned with restricted state-transition systems
- representation is usually restricted to propositional STRIPS
- neoclassical vs. classical planning
  - classical planning: search space consists of nodes containing partial plans
  - neoclassical planning: nodes can be seen as sets of partial plans
- resulted in significant speed-up and revival of planning research

Overview

- The Propositional Representation
- The Planning-Graph Structure
- The Graphplan Algorithm
Classical Representations

- **propositional representation**
  - world state is set of propositions
  - action consists of precondition propositions, propositions to be added and removed

- **STRIPS representation**
  - like propositional representation, but first-order literals instead of propositions

- **state-variable representation**
  - state is tuple of state variables \( \{x_1, \ldots, x_n\} \)
  - action is partial function over states

Propositional Planning Domains

- Let \( L = \{p_1, \ldots, p_n\} \) be a finite set of proposition symbols. A propositional planning domain on \( L \) is a restricted state-transition system \( \Sigma = (S, A, \gamma) \) such that:
  - \( S \subseteq 2^L \), i.e. each state \( s \) is a subset of \( L \)
  - \( A \subseteq 2^L \times 2^L \times 2^L \), i.e. each action \( a \) is a triple \((\text{precond}(a), \text{effects}^-(a), \text{effects}^+(a))\) where \( \text{effects}^-(a) \) and \( \text{effects}^+(a) \) must be disjoint
  - \( \gamma : S \times A \rightarrow 2^L \) where
    - \( \gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a) \) if \( \text{precond}(a) \subseteq s \)
    - \( \gamma(s, a) = \text{undefined} \) otherwise
  - \( S \) is closed under \( \gamma \)
DWR Example: State Space

The Graphplan Planner 7

DWR Example: Propositional States

- \( L = \{ \text{onpallet}, \text{onrobot}, \text{holding}, \text{at1}, \text{at2} \} \)
- \( S = \{ s_0, \ldots, s_5 \} \)
  - \( s_0 = \{ \text{onpallet}, \text{at2} \} \)
  - \( s_1 = \{ \text{holding}, \text{at2} \} \)
  - \( s_2 = \{ \text{onpallet}, \text{at1} \} \)
  - \( s_3 = \{ \text{holding}, \text{at1} \} \)
  - \( s_4 = \{ \text{onrobot}, \text{at1} \} \)
  - \( s_5 = \{ \text{onrobot}, \text{at2} \} \)

The Graphplan Planner 8
### DWR Example: Propositional Actions

<table>
<thead>
<tr>
<th>$a$</th>
<th>precond($a$)</th>
<th>effects$^-(a)$</th>
<th>effects$^+(a)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>take</td>
<td>{onpallet}</td>
<td>{onpallet}</td>
<td>{holding}</td>
</tr>
<tr>
<td>put</td>
<td>{holding}</td>
<td>{holding}</td>
<td>{onpallet}</td>
</tr>
<tr>
<td>load</td>
<td>{holding,at1}</td>
<td>{holding}</td>
<td>{onrobot}</td>
</tr>
<tr>
<td>unload</td>
<td>{onrobot,at1}</td>
<td>{onrobot}</td>
<td>{holding}</td>
</tr>
<tr>
<td>move1</td>
<td>{at2}</td>
<td>{at2}</td>
<td>{at1}</td>
</tr>
<tr>
<td>move2</td>
<td>{at1}</td>
<td>{at1}</td>
<td>{at2}</td>
</tr>
</tbody>
</table>

### DWR Example: Propositional State Transitions

<table>
<thead>
<tr>
<th></th>
<th>$s_0$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$s_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>take</td>
<td>$s_1$</td>
<td></td>
<td>$s_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>put</td>
<td></td>
<td>$s_0$</td>
<td></td>
<td>$s_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td></td>
<td></td>
<td>$s_4$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unload</td>
<td></td>
<td></td>
<td></td>
<td>$s_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>move1</td>
<td></td>
<td>$s_0$</td>
<td></td>
<td>$s_1$</td>
<td></td>
<td>$s_4$</td>
</tr>
<tr>
<td>move2</td>
<td>$s_2$</td>
<td></td>
<td>$s_3$</td>
<td></td>
<td></td>
<td>$s_5$</td>
</tr>
</tbody>
</table>
Propositional Planning Problems

• A propositional planning problem is a triple \( P=(\Sigma,s_i,g) \) where:
  - \( \Sigma=(S,A,\gamma) \) is a propositional planning domain on \( L=\{p_1,\ldots,p_n\} \)
  - \( s_i \in S \) is the initial state
  - \( g \subseteq L \) is a set of goal propositions that define the set of goal states \( S_g=\{s \in S \mid g \subseteq s\} \)

DWR Example: Propositional Planning Problem

• \( \Sigma \): propositional planning domain for DWR domain
• \( s_i \): any state
  - example: initial state = \( s_0 \in S \)
• \( g \): any subset of \( L \)
  - example: \( g=\{\text{onrobot,at2}\} \), i.e. \( S_g=\{s_5\} \)
Classical Plans

- A plan is any sequence of actions \( \pi = \langle a_1, \ldots, a_k \rangle \), where \( k \geq 0 \).
  - The length of plan \( \pi \) is \( |\pi| = k \), the number of actions.
  - If \( \pi_1 = \langle a_1, \ldots, a_k \rangle \) and \( \pi_2 = \langle a'_1, \ldots, a'_j \rangle \) are plans, then their concatenation is the plan \( \pi_1 \cdot \pi_2 = \langle a_1, \ldots, a_k, a'_1, \ldots, a'_j \rangle \).
  - The extended state transition function for plans is defined as follows:
    - \( \gamma(s, \pi) = s \) if \( k = 0 \) (\( \pi \) is empty)
    - \( \gamma(s, \pi) = \gamma(\gamma(s, a_1), \langle a_2, \ldots, a_k \rangle) \) if \( k > 0 \) and \( a_1 \) applicable in \( s \)
    - \( \gamma(s, \pi) = \text{undefined} \) otherwise

Classical Solutions

- Let \( \mathcal{P} = (\Sigma, s_i, g) \) be a propositional planning problem. A plan \( \pi \) is a solution for \( \mathcal{P} \) if \( g \subseteq \gamma(s_i, \pi) \).
  - A solution \( \pi \) is redundant if there is a proper subsequence of \( \pi \) is also a solution for \( \mathcal{P} \).
  - \( \pi \) is minimal if no other solution for \( \mathcal{P} \) contains fewer actions than \( \pi \).
### DWR Example: Plans and Solutions

| plan $\pi$                      | $|\pi|$ | $\gamma(s,\pi)$ | sol. | red. | min. |
|--------------------------------|---------|-----------------|------|------|------|
| {}                             | 0       | $s_0$           | no   | -    | -    |
| $\langle \text{move2}, \text{move2} \rangle$ | 2       | undef.          | no   | -    | -    |
| $\langle \text{take}, \text{move1} \rangle$ | 2       | $s_3$           | no   | -    | -    |
| $\langle \text{take}, \text{move1}, \text{put}, \text{move2}, \text{take}, \text{move1}, \text{load}, \text{move2} \rangle$ | 8       | $s_5$           | yes  | yes  | no   |
| $\langle \text{take}, \text{move1}, \text{load}, \text{move2} \rangle$ | 4       | $s_5$           | yes  | no   | yes  |
| $\langle \text{move1}, \text{take}, \text{load}, \text{move2} \rangle$ | 4       | $s_5$           | yes  | no   | yes  |

### Reachable Successor States

- The successor function $\Gamma^m : 2^S \rightarrow 2^S$ for a propositional domain $\Sigma = (S, A, \gamma)$ is defined as:
  - $\Gamma(s) = \{ \gamma(s, a) | a \in A \text{ and } a \text{ applicable in } s \}$ for $s \in S$
  - $\Gamma(\{s_1, \ldots, s_n\}) = \bigcup(k \in \{1, \ldots, n\}) \Gamma(s_k)$
  - $\Gamma^0(\{s_1, \ldots, s_p\}) = \{s_1, \ldots, s_p\}$
  - $\Gamma^m(\{s_1, \ldots, s_p\}) = \Gamma(\Gamma^{m-1}(\{s_1, \ldots, s_p\}))$

- The transitive closure of $\Gamma$ defines the set of all reachable states:
  - $\Gamma^\ast(s) = \bigcup(k \in \{0, \ldots\}) \Gamma^k(\{s\})$ for $s \in S$
Relevant Actions and Regression Sets

• Let $\mathcal{P} = (\Sigma, s, g)$ be a propositional planning problem. An action $a \in A$ is relevant for $g$ if
  • $g \cap \text{effects}^+(a) \neq \emptyset$ and
  • $g \cap \text{effects}^-(a) = \emptyset$.

• The regression set of $g$ for a relevant action $a \in A$ is:
  • $\gamma^{-1}(g, a) = (g - \text{effects}^+(a)) \cup \text{precond}(a)$
  • note: $\gamma(s, a) \in S_g$ iff $\gamma^{-1}(g, a) \subseteq s$

Regression Function

• The regression function $\Gamma^{-m}$ for a propositional domain $\Sigma = (S, A, \gamma)$ on $L$ is defined as:
  • $\Gamma^{-1}(g) = \{ \gamma^{-1}(g, a) \mid a \in A \text{ is relevant for } g \}$ for $g \in 2^L$
  • $\Gamma^0(g_1, \ldots, g_n) = \{ g_1, \ldots, g_n \}$
  • $\Gamma^{-1}(g_1, \ldots, g_n) = \bigcup_{k \in [1,n]} \Gamma^{-1}(g_k)$
  • $\Gamma^{-m}(g_1, \ldots, g_n) = \Gamma^{-1}(\Gamma^{-m-1}(\{g_1, \ldots, g_n\}))$

• The transitive closure of $\Gamma^{-1}$ defines the set of all regression sets:
  • $\Gamma^c(g) = \bigcup_{k \in [0, \infty]} \Gamma^{-k}(\{g\})$ for $g \in 2^L$
Statement of a Propositional Planning Problem

- A statement of a propositional planning problem is a triple $P = (A, s_i, g)$ where:
  - $A$ is a set of actions in an appropriate propositional planning domain $\Sigma = (S, A, \gamma)$ on $L$
  - $s_i$ is the initial state in an appropriate propositional planning problem $P = (\Sigma, s_i, g)$
  - $g$ is a set of goal propositions in the same propositional planning problem $P$

Example: Ambiguity in Statement of a Planning Problem

- statement: $P = (\{a_1\}, s_i, g)$ where $a_1 = (\{p_1\}, \{p_1, p_2\})$, $s_i = \{p_1\}$, and $g = \{p_2\}$

- $\Sigma_1 =$
  - $\{\{p_1\}, \{p_2\}\}$
  - $\{a_1\}$
  - $((\{p_1\}, a_1) \rightarrow \{p_2\})$ on $L_1 = \{p_1, p_2\}$

- $\Sigma_2 =$
  - $\{\{p_1\}, \{p_2\}, \{p_1, p_3\}, \{p_2, p_3\}\}$
  - $\{a_1\}$
  - $((\{p_1\}, a_1) \rightarrow \{p_2\}, ((\{p_1, p_3\}, a_1) \rightarrow \{p_2, p_3\})$ on $L_2 = \{p_1, p_2, p_3\}$
Statement Ambiguity

- **Proposition**: Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two propositional planning problems that have the same statement. Then both, $\mathcal{P}_1$ and $\mathcal{P}_2$, have
  - the same set of reachable states $\Gamma^*(s_j)$ and
  - the same set of solutions.

Properties of the Propositional Representation

- **Expressiveness**: For every propositional planning domain there is a corresponding state-transition system, but what about vice versa?
- **Conciseness**: propositional action representation is concise because it does not mention what does not change
- **Consistency**: not every assignment of truth values to propositions must correspond to a state in the underlying state-transition system
Grounding a STRIPS Planning Problem

- Let $P=(O,s_i,g)$ be the statement of a STRIPS planning problem and $C$ the set of all the constant symbols that are mentioned in $s_i$. Let $\text{ground}(O)$ be the set of all possible instantiations of operators in $O$ with constant symbols from $C$ consistently replacing variables in preconditions and effects.
- Then $P'=(\text{ground}(O),s_i,g)$ is a statement of a STRIPS planning problem and $P'$ has the same solutions as $P$.

Translation: Propositional Representation to Ground STRIPS

- Let $P=(A,s_i,g)$ be a statement of a propositional planning problem. In the actions $A$:
  - replace every action $(\text{precond}(a), \text{effects}^-(a), \text{effects}^+(a))$ with an operator $o$ with
    - some unique name($o$),
    - $\text{precond}(o) = \text{precond}(a)$, and
    - $\text{effects}(o) = \text{effects}^+(a) \cup \{\neg p \mid p \in \text{effects}^-(a)}$. 

Translation: Ground STRIPS to Propositional Representation

- Let \( P = (O, s, g) \) be a ground statement of a classical planning problem.
  - In the operators \( O \), in the initial state \( s \), and in the goal \( g \) replace every atom \( P(v_1, \ldots, v_n) \) with a propositional atom \( P_{v_1, \ldots, v_n} \).
  - In every operator \( o \): for all \( \neg p \) in \( \text{precond}(o) \), replace \( \neg p \) with \( p' \).
    - if \( p \) in \( \text{effects}(o) \), add \( \neg p' \) to \( \text{effects}(o) \).
    - if \( \neg p \) in \( \text{effects}(o) \), add \( p' \) to \( \text{effects}(o) \).
  - In the goal replace \( \neg p \) with \( p' \).
  - For every operator \( o \) create an action \( (\text{precond}(o), \text{effects}^{-}(a), \text{effects}^{+}(a)) \).

Overview

- The Propositional Representation
  - The Planning-Graph Structure
  - The Graphplan Algorithm
Example: Simplified DWR Problem

- robots can load and unload autonomously
- locations may contain unlimited number of robots and containers
- problem: swap locations of containers

Simplified DWR Problem: STRIPS Actions

- move($r,l,l'$)
  - precond: at($r,l$), adjacent($l,l'$)
  - effects: at($r,l'$), ¬at($r,l$)
- load($c,r,l$)
  - precond: at($r,l$), in($c,l$), unloaded($r$)
  - effects: loaded($r,c$), ¬in($c,l$), ¬unloaded($r$)
- unload($c,r,l$)
  - precond: at($r,l$), loaded($r,c$)
  - effects: unloaded($r$), in($c,l$), ¬loaded($r,c$)
**Simplified DWR Problem: State Proposition Symbols**

- **robots:**
  - \( r1 \) and \( r2 \): \( \text{at(robr,loc1)} \) and \( \text{at(robr,loc2)} \)
  - \( q1 \) and \( q2 \): \( \text{at(robq,loc1)} \) and \( \text{at(robq,loc2)} \)
  - \( ur \) and \( uq \): \( \text{unloaded(robr)} \) and \( \text{unloaded(robq)} \)

- **containers:**
  - \( a1, a2, ar, \) and \( aq \): \( \text{in(conta,loc1)} \), \( \text{in(conta,loc2)} \), \( \text{loaded(conta,robr)} \), and \( \text{loaded(conta,robq)} \)
  - \( b1, b2, br, \) and \( bq \): \( \text{in(contb,loc1)} \), \( \text{in(contb,loc2)} \), \( \text{loaded(contb,robr)} \), and \( \text{loaded(contb,robq)} \)

- **initial state:** \( \{r1, q2, a1, b2, ur, uq\} \)

**Simplified DWR Problem: Action Symbols**

- **move actions:**
  - \( Mr12 \): \( \text{move(robr,loc1,loc2)} \), \( Mr21 \):
    \( \text{move(robr,loc2,loc1)} \), \( Mq12 \): \( \text{move(robq,loc1,loc2)} \), \( Mq21 \): \( \text{move(robq,loc2,loc1)} \)

- **load actions:**
  - \( Lar1 \): \( \text{load(conta,robr,loc1)} \); \( Lar2, Laq1, Laq2, Lar1, Lbr2, Lbq1, \) and \( Lbq2 \) correspondingly

- **unload actions:**
  - \( Uar1 \): \( \text{unload(conta,robr,loc1)} \); \( Uar2, Uaq1, Uaq2, Uar1, Ubr2, Ubq1, \) and \( Ubq2 \) correspondingly
Solution Existence

• **Proposition**: A propositional planning problem \( P = (\Sigma, s_i, g) \) has a solution iff 
  \( S_g \cap \Gamma^>(\{s_i\}) \neq \emptyset \).

• **Proposition**: A propositional planning problem \( P = (\Sigma, s_i, g) \) has a solution iff 
  \( \exists s \in \Gamma^<(\{g\}) : s \subseteq s_i \).

Reachability Tree

• tree structure, where:
  • root is initial state \( s_i \)
  • children of node \( s \) are \( \Gamma(\{s\}) \)
  • arcs are labelled with actions
• all nodes in reachability tree are \( \Gamma^>(\{s_i\}) \)
  • all nodes to depth \( d \) are \( \Gamma^d(\{s_i\}) \)
  • solves problems with up to \( d \) actions in solution

• problem: \( O(k^d) \) nodes; 
  \( k = \) applicable actions per state
DWR Example: Reachability Tree

Planning Graph: Nodes

- layered directed graph $G=(N,E)$:
  - $N = P_0 \cup A_1 \cup P_1 \cup A_2 \cup P_2 \cup \ldots$
    - state proposition layers: $P_0, P_1, \ldots$
    - action layers: $A_1, A_2, \ldots$
- first proposition layer $P_0$:
  - propositions in initial state $s_i$: $P_0 = s_i$
- action layer $A_j$:
  - all actions $a$ where: $\text{precond}(a) \subseteq P_{j-1}$
- proposition layer $P_j$:
  - all propositions $p$ where: $p \in P_{j-1}$ or $\exists a \in A_j: p \in \text{effects}^+(a)$
Planning Graph: Arcs

- from proposition $p \in P_{j-1}$ to action $a \in A_j$:
  - if: $p \in \text{precond}(a)$
- from action $a \in A_j$ to layer $p \in P_j$:
  - positive arc if: $p \in \text{effects}^+(a)$
  - negative arc if: $p \in \text{effects}^-(a)$

- no arcs between other layers
Reachability in the Planning Graph

- reachability analysis:
  - if a goal $g$ is reachable from initial state $s_i$
  - then there will be a proposition layer $P_g$ in the planning graph such that $g \subseteq P_g$

- necessary condition, but not sufficient
- low complexity:
  - planning graph is of polynomial size and
  - can be computed in polynomial time

Independent Actions: Examples

- Mr12 and Lar1:
  - cannot occur together
  - Mr12 deletes precondition $r_1$ of Lar1

- Mr12 and Mr21:
  - cannot occur together
  - Mr12 deletes positive effect $r_1$ of Mr21

- Mr12 and Mq21:
  - may occur in same action layer
Independent Actions

- Two actions $a_1$ and $a_2$ are independent iff:
  - $\text{effects}^{-}(a_1) \cap (\text{precond}(a_2) \cup \text{effects}^{+}(a_2)) = \emptyset$
  - $\text{effects}^{-}(a_2) \cap (\text{precond}(a_1) \cup \text{effects}^{+}(a_1)) = \emptyset$.
- A set of actions $\pi$ is independent iff every pair of actions $a_1, a_2 \in \pi$ is independent.

Pseudo Code: independent

function independent($a_1, a_2$)
  for all $p \in \text{effects}^{-}(a_1)$
    if $p \in \text{precond}(a_2)$ or $p \in \text{effects}^{+}(a_2)$ then
      return false
  for all $p \in \text{effects}^{-}(a_2)$
    if $p \in \text{precond}(a_1)$ or $p \in \text{effects}^{+}(a_1)$ then
      return false
  return true
Applying Independent Actions

- A set $\pi$ of independent actions is *applicable* to a state $s$ iff $\bigcup_{a \in \pi} \text{precond}(a) \subseteq s$.
- The result of applying the set $\pi$ in $s$ is defined as:
  $$\gamma(s, \pi) = (s - \text{effects}^{-}(\pi)) \cup \text{effects}^{+}(\pi),$$
  where:
  - $\text{precond}(\pi) = \bigcup_{a \in \pi} \text{precond}(a)$,
  - $\text{effects}^{+}(\pi) = \bigcup_{a \in \pi} \text{effects}^{+}(a)$, and
  - $\text{effects}^{-}(\pi) = \bigcup_{a \in \pi} \text{effects}^{-}(a)$.

Execution Order of Independent Actions

- **Proposition**: If a set $\pi$ of independent actions is applicable in state $s$ then, for any permutation $\langle a_1, \ldots, a_k \rangle$ of the elements of $\pi$:
  - the sequence $\langle a_1, \ldots, a_k \rangle$ is applicable to $s$, and
  - the state resulting from the application of $\pi$ to $s$ is the same as from the application of $\langle a_1, \ldots, a_k \rangle$, i.e.:
  $$\gamma(s, \pi) = \gamma(s, \langle a_1, \ldots, a_k \rangle).$$
Layered Plans

- Let $P = (A, s, g)$ be a statement of a propositional planning problem and $G = (N, E)$, $N = P_0 \cup A_1 \cup P_1 \cup A_2 \cup P_2 \cup \ldots$, the corresponding planning graph.
- A layered plan over $G$ is a sequence of sets of actions: $\prod = \langle \pi_1, \ldots, \pi_k \rangle$ where:
  - $\pi_i \subseteq A_i \subseteq A$,
  - $\pi_i$ is applicable in state $P_{i-1}$, and
  - the actions in $\pi_i$ are independent.

Layered Solution Plan

- A layered plan $\prod = \langle \pi_1, \ldots, \pi_k \rangle$ is a solution to a planning problem $P = (A, s, g)$ iff:
  - $\pi_1$ is applicable in $s$,
  - for $j \in \{2, \ldots, k\}$, $\pi_j$ is applicable in state $\gamma(\ldots \gamma(\gamma(s, \pi_1), \pi_2), \ldots, \pi_{j-1})$, and
  - $g \subseteq \gamma(\ldots \gamma(\gamma(s, \pi_1), \pi_2), \ldots, \pi_k)$.
Execution Order in Layered Solution Plans

- **Proposition:** If \( \Pi = (\pi_1, \ldots, \pi_k) \) is a solution to a planning problem \( P=(A,s_i,g) \), then:
  - a sequence of actions corresponding to any permutation of the elements of \( \pi_1 \),
  - followed by a sequence of actions corresponding to any permutation of the elements of \( \pi_2 \),
  - …
  - followed by a sequence of actions corresponding to any permutation of the elements of \( \pi_k \)

is a path from \( s_i \) to a goal state.

Problem: Dependent Propositions: Example

- \( r2 \) and \( ar \):
  - \( r2 \): positive effect of Mr12
  - \( ar \): positive effect of Lar1
  - but: Mr12 and Lar1 not independent
  - hence: \( r2 \) and \( ar \) incompatible in \( P_1 \)
- \( r1 \) and \( r2 \):
  - positive and negative effects of same action: Mr12
  - hence: \( r1 \) and \( r2 \) incompatible in \( P_1 \)
No-Operation Actions

- No-Op for proposition \( p \):
  - name: \( Ap \)
  - precondition: \( p \)
  - effect: \( p \)
- \( r1 \) and \( r2 \):
  - \( r1 \): positive effect of \( Ar1 \)
  - \( r2 \): positive effect of \( Mr12 \)
  - but: \( Ar1 \) and \( Mr12 \) not independent
  - hence: \( r1 \) and \( r2 \) incompatible in \( P_1 \)
- only one incompatibility test

Mutex Propositions

- Two propositions \( p \) and \( q \) in proposition layer \( P_j \) are mutex (mutually exclusive) if:
  - every action in the preceding action layer \( A_j \) that has \( p \) as a positive effect (incl. no-op actions) is mutex with every action in \( A_j \) that has \( q \) as a positive effect, and
  - there is no single action in \( A_j \) that has both, \( p \) and \( q \), as positive effects.
- notation: \( \mu P_j = \{ (p,q) | p,q \in P_j \text{ are mutex} \} \)
**Pseudo Code: mutex for Propositions**

```plaintext
function mutex(p1, p2, μA_j)
    for all a_1 ∈ p1.producers()
        for all a_2 ∈ p2.producers()
            if (a_1, a_2) ∉ μA_j then
                return false
            end if
        end for
    end for
    return true
end function
```

**Mutex Actions: Example**

- r1 and r2 are mutex in $P_1$
- r1 is precondition for Lar1 in $A_2$
- r2 is precondition for Mr21 in $A_2$
- hence: Lar1 and Mr21 are mutex in $A_2$
Mutex Actions

- Two actions $a_1$ and $a_2$ in action layer $A_j$ are mutex if:
  - $a_1$ and $a_2$ are dependent, or
  - a precondition of $a_1$ is mutex with a precondition of $a_2$.
- notation:
  $\mu A_j = \{ (a_1, a_2) \mid a_1, a_2 \in A_j \text{ are mutex} \}$

Pseudo Code: mutex for Actions

```pseudocode
function mutex(a_1, a_2, P)
    if ¬independent(a_1, a_2) then
        return true
    for all $p_1 \in \text{precond}(a_1)$
        for all $p_2 \in \text{precond}(a_2)$
            if $(p_1, p_2) \in P$ then return true
    return false
```
Decreasing Mutex Relations

- **Proposition**: If \( p,q \in P_{j-1} \) and \((p,q) \notin \mu P_{j-1}\) then \((p,q) \notin \mu P_j\).
  - **Proof**:
    - if \( p,q \in P_{j-1} \) then \( Ap,Aq \in A_j \)
    - if \((p,q) \notin \mu P_{j-1}\) then \((Ap,Aq) \notin \mu A_j\)
    - since \( Ap,Aq \in A_j \) and \((Ap,Aq) \notin \mu A_j\), \((p,q) \notin \mu P\) must hold
- **Proposition**: If \( a_1,a_2 \in A_{j-1} \) and \((a_1,a_2) \notin \mu A_{j-1}\) then \((a_1,a_2) \notin \mu A_j\).
  - **Proof**:
    - if \( a_1,a_2 \in A_{j-1} \) and \((a_1,a_2) \notin \mu A_{j-1}\) then
      - \( a_1 \) and \( a_2 \) are independent and
      - their preconditions in \( P_{j-1} \) are not mutex
    - both properties remain true for \( P_j \)
    - hence: \( a_1,a_2 \in A_j \) and \((a_1,a_2) \notin \mu A_j\)

Removing Impossible Actions

- Actions with mutex preconditions \( p \) and \( q \) are impossible
  - example: preconditions \( r_2 \) and \( ar \) of \( Uar2 \) in \( A_2 \) are mutex
  - can be removed from the graph
    - example: remove \( Uar2 \) from \( A_2 \)

The Graphplan Planner 54
Reachability in Planning Graphs

- **Proposition**: Let $P = (A, s_i, g)$ be a propositional planning problem and $G = (N, E)$, $N = P_0 \cup A_1 \cup P_1 \cup A_2 \cup P_2 \cup \ldots$, the corresponding planning graph. If
  - $g$ is reachable from $s_i$
  then
  - there is a proposition layer $P_g$ such that
    - $g \subseteq P_g$ and
    - $\neg \exists g_1, g_2 \in g : (g_1, g_2) \in \mu_{P_g}$.

Overview

- The Propositional Representation
- The Planning-Graph Structure
  - The Graphplan Algorithm
The Graphplan Algorithm: Basic Idea

- expand the planning graph, one action layer and one proposition layer at a time
- from the first graph for which $P_g$ is the last proposition layer such that
  - $g \subseteq P_g$ and
  - $\neg \exists g_1, g_2 \in g: (g_1, g_2) \in \mu P_g$
- search backwards from the last (proposition) layer for a solution

Planning Graph Data Structure

- $k$-th planning graph $G_k$:
  - nodes $N$:
    - array of proposition layers $P_0 \ldots P_k$
      - proposition layer $j$: set of proposition symbols
    - array of action layers $A_1 \ldots A_k$
      - proposition layer $j$: set of action symbols
  - edges $E$:
    - precondition links: $pre_j \subseteq P_j \times A_j$, $j \in \{1 \ldots k\}$
    - positive effect links: $e^+_j \subseteq A_j \times P_j$, $j \in \{1 \ldots k\}$
    - negative effect links: $e^-_j \subseteq A_j \times P_j$, $j \in \{1 \ldots k\}$
    - proposition mutex links: $\mu A_j \subseteq A_j \times A_j$, $j \in \{1 \ldots k\}$
    - action mutex links: $\mu P_j \subseteq P_j \times P_j$, $j \in \{1 \ldots k\}$
Pseudo Code: expand

function expand($G_{k-1}$)

$A_k \leftarrow \{ a \in A \mid \text{precond}(a) \subseteq P_{k-1} \}$ and

$\{(p_1, p_2) \mid p_1, p_2 \in \text{precond}(a) \} \cap \mu P_{k-1} = \{\} \}

$\mu A_k \leftarrow \{(a_1, a_2) \mid a_1, a_2 \in A_k, a_1 \neq a_2, \text{ and mutex}(a_1, a_2, \mu P_{k-1})\}

$P_k \leftarrow \{ p \mid \exists a \in A_k : p \in \text{effects}^+(a) \}

$\mu P_k \leftarrow \{(p_1, p_2) \mid p_1, p_2 \in P_k, p_1 \neq p_2, \text{ and mutex}(p_1, p_2, \mu A_k)\}

for all $a \in A_k$

$pre_k \leftarrow pre_k \cup \{(p \mid p \in P_{k-1} \text{ and } p \in \text{precond}(a)) \times a\}

$e_k^+ \leftarrow e_k^+ \cup (a \times \{p \mid p \in P_k \text{ and } p \in \text{effects}^+(a)\})

$e_k^- \leftarrow e_k^- \cup (a \times \{p \mid p \in P_k \text{ and } p \in \text{effects}^-(a)\})

Planning Graph Complexity

- **Proposition:** The size of a planning graph up to level $k$ and the time required to expand it to that level are polynomial in the size of the planning problem.

- **Proof:**
  - problem size: $n$ propositions and $m$ actions
  - $|P| \leq n$ and $|A| \leq n + m$ (incl. no-op actions)
  - algorithms for generating each layer and all link types are polynomial in size of layer
Fixed-Point Levels

- A fixed-point level in a planning graph $G$ is a level $\kappa$ such that for all $i, i > \kappa$, level $i$ of $G$ is identical to level $\kappa$, i.e. $P_i = P_\kappa$, $\mu P_i = \mu P_\kappa$, $A_i = A_\kappa$, and $\mu A_i = \mu A_\kappa$.

- **Proposition**: Every planning graph $G$ has a fixed-point level $\kappa$, which is the smallest $k$ such that $|P_k| = |P_{k+1}|$ and $|\mu P_k| = |\mu P_{k+1}|$.

- **Proof**:
  - $P_i$ grows monotonically and $\mu P_i$ shrinks monotonically
  - $A_i$ and $P_i$ only depend on $P_{i-1}$ and $\mu P_{i-1}$

Searching the Planning Graph

- **general idea**:
  - search backwards from the last proposition layer $P_k$ in the current graph
  - let $g$ be the set of goal propositions that need to be achieved at a given proposition layer $P_j$ (initially the last layer)
  - find a set of actions $\pi_j \subseteq A_j$ such that these actions are not mutex and together achieve $g$
  - take the union of the preconditions of $\pi_j$ as the new goal set to be achieved in proposition layer $P_{j-1}$
Planning Graph Search Example

Planning Graph as AND/OR-Graph

- OR-nodes:
  - nodes in proposition layers
  - links to actions that support the propositions

- AND-nodes:
  - nodes in action layers
  - $k$-connectors all preconditions of the action

- search:
  - $AO^*$ not best algorithm because it does not exploit layered structure
Repeated Sub-Goals

The Graphplan Planner

The nogood Table

- *nogood* table (denoted \( \nabla \)) for planning graph up to layer \( k \):
  - array of \( k \) sets of sets of goal propositions
    - inner set: one combination of propositions that cannot be achieved
    - outer set: all combinations that cannot be achieved (at that layer)

- before searching for set \( g \) in \( P_j \):
  - check whether \( g \in \nabla(j) \)
- when search for set \( g \) in \( P_j \) has failed:
  - add \( g \) to \( \nabla(j) \)
### Pseudo Code: extract

```plaintext
function extract(G,g,i)
    if i=0 then return ∅
    if g∈∇(i) then return failure
    Π ← gpSearch(G,g,{},i)
    if Π≠failure then return Π
    ∇(i) ← ∇(i) + g
    return failure
```

### Pseudo Code: gpSearch

```plaintext
function gpSearch(G,g,π,i)
    if g={} then
        Π ← extract(G,∪a∈π precond(a),i-1)
        if Π=failure then return failure
        return Π⊙⟨π⟩
    p ← g.selectOne()
    resolvers ← {a∈Ai | p∈effects+{a} and ¬∃a′∈π: (a,a′)∈μAi}
    if resolvers={} then return failure
    a ← resolvers.chooseOne()
    return gpSearch(G,g-effects+(a),π+a,i)
```
Pseudo Code: graphplan

function graphplan(A,s,g)
    i ← 0;  ; P_0 ← s; G ← (P_0,();
    while (g ∉ P, or g ∉ P) and ¬fixedPoint(G) do
        i ← i + 1; expand(G)
        if g ∉ P, or g ∉ P then return failure
        η ← fixedPoint(G) ? |∇(κ)| : 0
        |[i] ← extract(G,g,i)
    while |[i]|=failure do
        i ← i + 1; expand(G)
        |[i] ← extract(G,g,i)
        if |[i]|=failure and fixedPoint(G) then
            if r|∇(κ)| then return failure
            η ← |∇(κ)|
        return |[i]

Graphplan Properties

- **Proposition**: The Graphplan algorithm is sound, complete, and always terminates.
  - It returns failure iff the given planning problem has no solution;
  - otherwise, it returns a layered plan |[i] that is a solution to the given planning problem.

- Graphplan is orders of magnitude faster than previous techniques!
Overview

- The Propositional Representation
- The Planning-Graph Structure
- The Graphplan Algorithm
- Planning-Graph Heuristics

Forward State-Space Search

- idea: apply standard search algorithms (breadth-first, depth-first, A*, etc.) to planning problem:
  - search space is subset of state space
  - nodes correspond to world states
  - arcs correspond to state transitions
  - path in the search space corresponds to plan
DWR Example State

goal: (and
  (in ca p2) (in cb q2) (in cc p2) (in cd q2) (in ce q2) (in cf q2))

Heuristics

• estimate distance to nearest goal state
  • number of unachieved goals (not admissible)
  • number of unachieved goals / max. number of positive effects per operator (admissible)

• example state (prev. slide):
  • actual goal distance: 35 actions
  • h(s) = 6
  • h(s) = 6 / 4
Finding Better Heuristics

- solve “relaxed” problem and use solution as heuristic
- planning heuristic:
  - planning problem: \( P = (O, s_i, g) \)
  - for \( p \in g \): \( \text{min-layer}(p) = \text{index of first proposition layer in planning graph that contains } p \)
  - admissible heuristic: \( \max(p \in g): \text{min-layer}(p) \)
  - not admissible: \( \sum(p \in g): \text{min-layer}(p) \)
- no need to compute mutex relations
- no need to re-compute planning graph for ground backward search

The FF Planner (Basics)

- heuristic
  - based on planning graph without negative effects
  - backward search possible in polynomial time
- search strategy
  - enforced hill-climbing: commit to first state with better f-value
Overview

- The Propositional Representation
- The Planning-Graph Structure
- The Graphplan Algorithm
- Planning-Graph Heuristics