Part V

Planning-Graph Techniques

Outline of Part VI

Introduction

- Planning-Graph techniques rely on classical planning representation
- These techniques introduce a new search space called Planning-Graph
- Planning-Graph techniques provide plan as a sequence of sets of actions
  - Plan-space produces plan as a partially ordered set of actions
  - State-space produce plan as a sequence of actions
  - $\Longrightarrow$ Planning-Graph is less expressive that Plan-space but more than State-space
- Planning-Graph approach rely on two interrelated ideas:
  1. Reachability analysis: addresses the issue of wether a state is reachable from some given state
  2. Disjunctive refinement: consists of addressing one or several flaws through a disjunctive resolvers

Reachability Trees

- The planning-graph structure provides a efficient way to estimate which set of propositions is possible reachable from a state s_0 with which actions

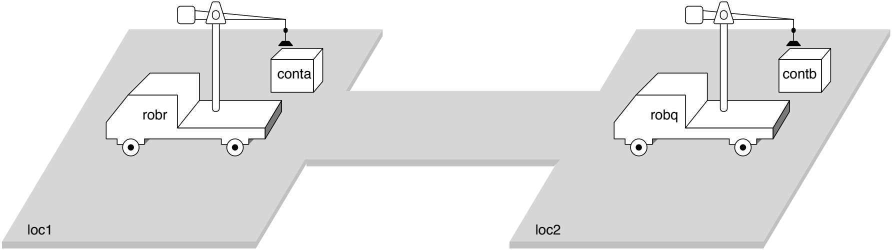
Definition (Reachability)

Given a set A of actions, a state s is reachable from some initial state s_0, if there is a sequence of actions in A that defines a path form s_0 to s.

- Reachability analysis consists in analysing which states can be reached from s_0 in some number of steps and how to reach them
  - can be used to defined heuristics in state-space planning
- Reachability can be computed exactly through a reachability tree that gives $\hat{\Gamma}(s_0)$, or it can be approximated though planning graph developped

Example: Reachability Trees

- Consider a simpled DWR domain with no piles and no cranes where robots can load and unload autonomously containers where locations locations can contain an unlimited number of robots



Example: Reachability Trees

Operators

move(r, l, l') *;; robot r at location l moves at a connected location l'*

precond: at(r, l), adjacent(l, l')
effects: at(r,l'), $\neg$at(r, l)

load(c, r, l) *;; robot r at location l loads container c*

precond: at(r, l), in(c,l), unloaded(r)
effects: loaded(r,c), $\neg$in(c, l), $\neg$unloaded(r)

unload(c, r, l) *;; robot r at location l unloads container c*

precond: at(r, l), loaded(r, c)
effects: unloaded(r), in(c,l), $\neg$loaded(r, c)

- Here the set of actions A has 20 actions corresponding to the operators move, load and unload

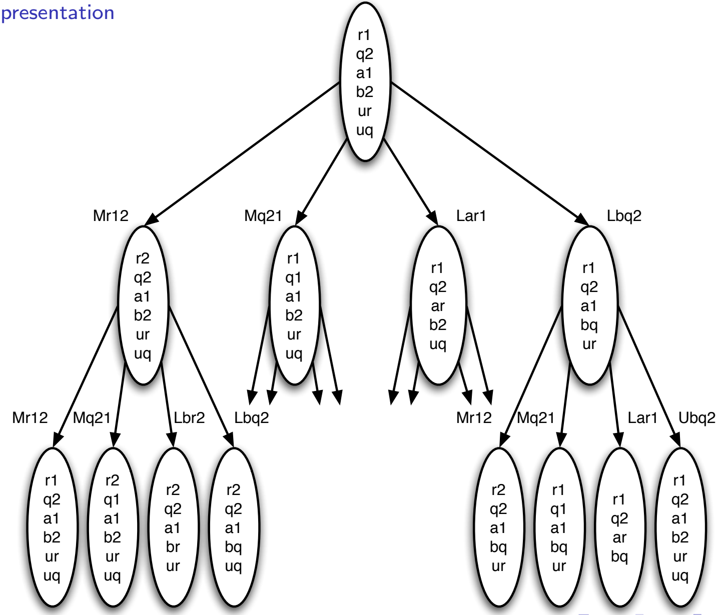
Example: Reachability Trees

Notation

- To simplify representation, let us denote atoms by propositional symbols:
  - r_1 and r_2 stand for at(robr,loc1) and at(robr,loc2)
  - q_1 and q_2 stand for at(robq,loc1) and at(robq,loc2)
  - a_1, a_2, a_r and a_q stand for int(conta,loc1), in(conta,loc2), loaded(conta,robr) and loaded(conta,robq)
  - b_1, b_2, b_r and b_q stand for int(contb,loc1), in(contb,loc2), loaded(contb,robr) and loaded(contb,robq)
- Let us also denote the 20 actions in A:
  - $Mr12$ is the action move(robr,loc1,loc2), $Mr21$ is the opposite, and $Mq12$ and $Mq21$ are the similar move action for robot robq
  - $Lar1$ is the action load(conta,robr,loc1) $Lar2$, $Laq1$ and $Laq2$ are the other load actions for conta in loc2 with contb. $Lbr1$, $Lbr2$, $Lbq1$ and $Lbq2$ are the load actions for contb
  - $Uar1$, $Uar2$, $Uaq1$, $Uaq2$, $Ubr1$, $Ubr2$, $Ubq1$, and $Ubq2$ are the unload actions

Example: Reachability Trees

Graphic representation



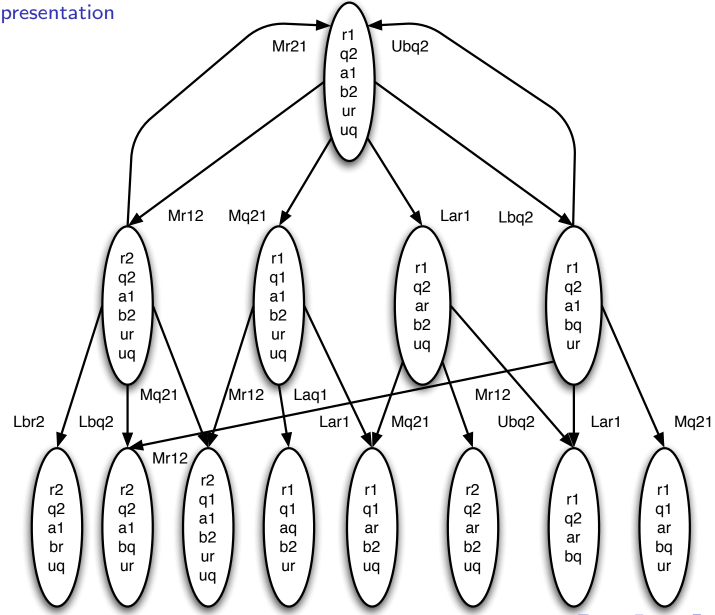
Reachability Trees

- A reachability tree is a tree T whose
  - Nodes are states of Σ
  - Edges corresponds to action of Σ
- The root node of T is the state s_0
- The children of a node s are all the state in $\Gamma(s)$
- A complete reachability tree from s_0 give $\hat{\Gamma}(s_0)$
- A reachability tree developed down to depth d solves all planning problems with s_0 and A, for every goal that is reachable in d of fewer actions:
  - a goal is reachable from s_0 in at most d steps iff it appears in some node of the tree
- The size of T blows up in $O(k^d)$, where k is the number of valid action per state
- Some nodes of T can be reached by different paths

  $\Longrightarrow$ reachability tree can be factorized into a graph

Example: Reachability graph

Graphic representation



Reachability with Planning Graphs

- A major contribution of Graphplan planner is a relaxation of the reachability analysis
- The approach provides an incomplete condition of reachability through a planning graph
  - A goal is reachable from s_0 *only if* it appears in some node of the planning graph : this is not a sufficient condition anymore
  - This weak reachability condition is compensated for a low complexity
  - The planning graph is of polynomial size and can be build in polynomial time in the size of the input

Reachability with Planning Graphs

Basic Idea

Basic Idea

The basic idea in a planning graph is to consider at every level of this structure not individual states but, to a first approximation, the *union* of sets of propositions in serveral states

Reachability with Planning Graphs

Basic Idea

Reachability tree	Planning Graph
• Actions branching out form a node are mutually exclusive	• Actions are consideres as inclusive disjunction from a node to the next that contains all the effects of the actions
• A node is associated with the proposition that necessarily hold for that node	• A node contains proposition that possibly hold at some point
• State is a consistent set of propositions	• The union of the sets of propositions for several states does not preserve consistency $\Longrightarrow$ solution is to keep track incompatible actions and propositions

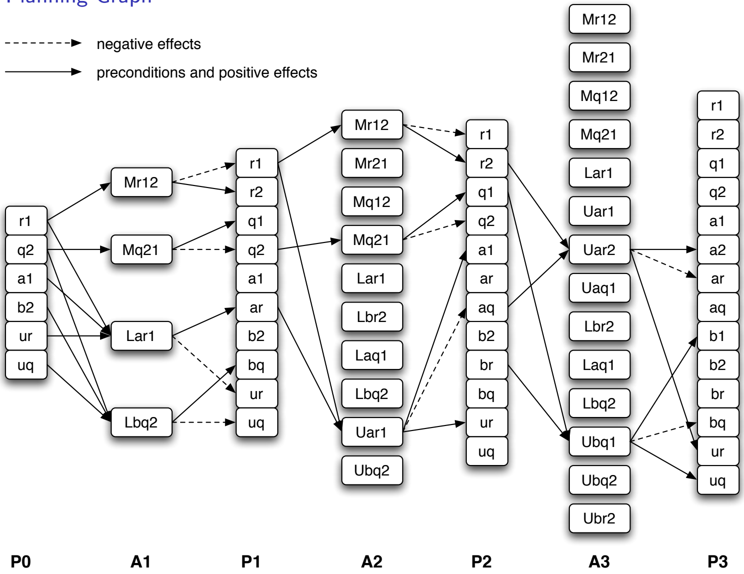
Reachability with Planning Graphs

Informal Definition

- A planning graph is a directed layered graph:
  - arcs are permitted only from one layer to the next
- Nodes in level 0 correspond to the set P_0 of propositions denoting the initial state s_0 of the planning problem
- Level 1 contains two layers:
  1. an action level A_1 that is the set of actions (ground instance of operators) whose preconditions are nodes in P_0
  2. a proposition level P_1 that is defined as the union of P_0 and the sets of positive effects of action in A_1
- An action node in A_1 is connected with :
  - a incoming precondition arcs from its preconditions in P_0
  - a outgoing arcs to its positive effects and to its negative effects in P_1
- Outgoing arcs are labeled positive or negative
  - Note that negative effects are not deleted from P_1, thus $P_0 \subseteq P_1$
- This process is pursued from one level to the next

Reachability with Planning Graphs

Exemple: Planning Graph



Reachability with Planning Graphs

Remarks

- In accordance with the idea of inclusive disjunction in A_i and the union of proposition in P_i, a plan associated to a planning graph is no longer a sequence of actions corresponding directly to a path in Σ
  - A plan Π is sequence of set of actions

$$\Pi = \langle \pi_1, \pi_2, \ldots, \pi_k \rangle$$

  - A plan is qualified as layered plan since it is is organized into levels corresponding to those of the planning graph with $\pi_i \subseteq A_i$
- The first level π_1 is a subset of indepedent action in A_1 that can be apply in any order to the initial state and can lead to a state that is a subset of P_1 and so forth until level k whose actions lead to a state meeting the goal

Independent Actions

Definition

Definition (Independent Actions)

Two actions (a, b) are independent iff:

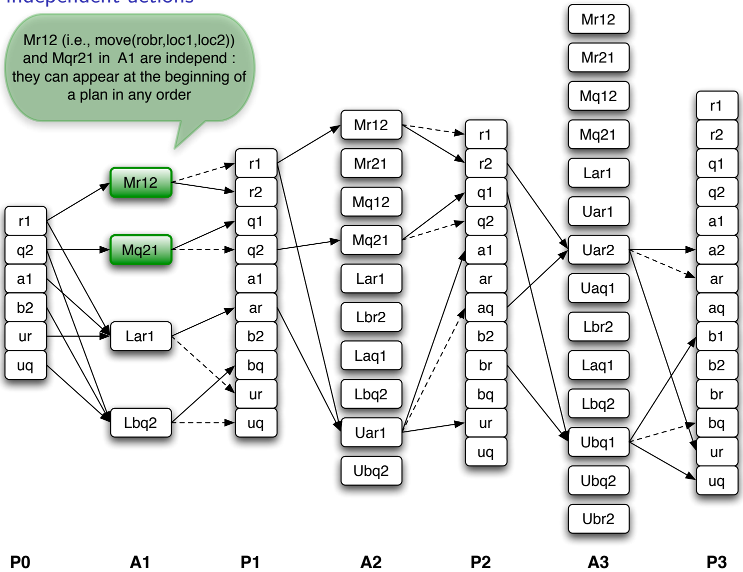
- $\text{effects}^{-}(a) \cap [\ \text{precond}(b) \cup \text{effect}^{+}(b)\ ] = \emptyset$ and
- $\text{effects}^{-}(b) \cap [\ \text{precond}(a) \cup \text{effect}^{+}(a)\ ] = \emptyset$ and

A set of actions π is independent when every pair of π is independent

- Conversely, two actions a and b are dependent if:
  - a deletes a precondition of b or
    - the ordering $a \prec b$ will not be permitted
  - a deletes a positive effect of b or
    - the resulting state will depend on their order
  - symetrically for negative effects of b with respect to a
    - b deletes a precondition on a positive effect of a

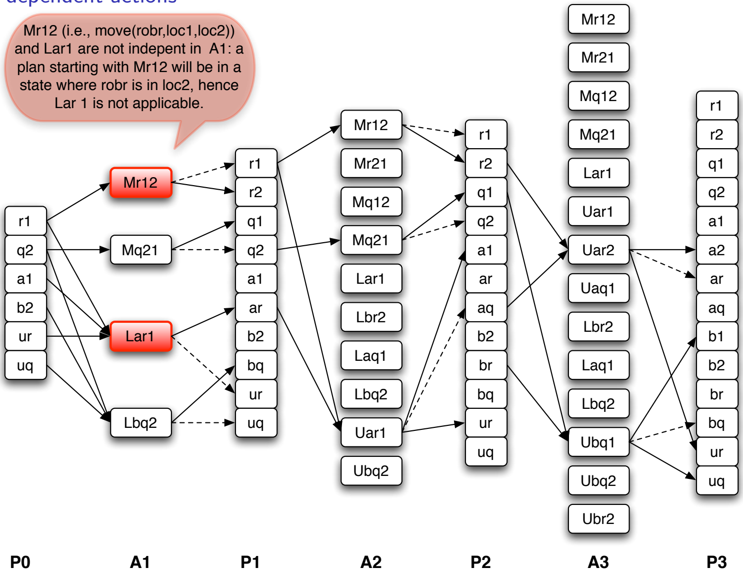
Independent Actions

Example: independent actions



Independent Actions

Example: dependent actions



Independent Actions

Remarks

1. The independence of action is not specific to a particular planning problem
2. It is intrinsic property of the actions of a domain that can be computed beforehand for all problems of that domain

Independent Actions

Applicable Actions

Definition (Actions Applicable)

A set π of independent actions is applicable to a state s iff $\text{precond}(\pi) \subseteq s$. The result of applying the set π to s is defined as:

$$\gamma(s, \pi) = (s - \text{effects}^-(\pi)) \cup \text{effects}^+(\pi)$$

where

- $\text{precond}(\pi) = \bigcup \{\text{precond}(a) \mid \forall a \in \pi\}$,
- $\text{effects}^+(\pi) = \bigcup \{\text{effects}^+(a) \mid \forall a \in \pi\}$, and
- $\text{effects}^-(\pi) = \bigcup \{\text{effects}^-(a) \mid \forall a \in \pi\}$.

Independent Actions

Applicable Actions' Set

Proposition (Applicable Actions' Set)

If a set π of independent actions is applicable to s then, for any permutation $\langle a_1, \ldots, a_k \rangle$ of the elements of π, the sequence $\langle a_1, \ldots, a_k \rangle$ is applicable to s, and the state resulting from the application of π to s is such that

$$\gamma(s, \pi) = \gamma(\ldots \gamma(\gamma(s, a_1), a_2), \ldots a_k)$$

Note

This proposition allow to go back to the standard semantics pf a plan in a state-transition system from the initial state to goal

Layered Plan

Definition

Definition (Layered Plan)

A layered plan is a sequence of set of actions. The layered plan $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$ is a solution to a problem (O, s_0, g) iff :

- each set $\pi \in \Pi$ is applicable to $\gamma(s_0, \pi_1, \ldots)$ and
- $g \subseteq \gamma(\ldots \gamma(\gamma(s, \pi_1), \pi_2), \ldots \pi_n)$.

Proposition (Layered Plan Concurrency)

If $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$ is a solution plan to a problem (O, s_0, g), then a sequence of actions coresponding to any permutation of the elements of π_1, followed by any permutation of π_2, ..., follow by any permutation of π_n is a path from s_0 to a goal state.

- This proposition follows directly the actions concurrency proposition.

Mutual Exclusion Relations

Overview

- The union of the sets of propositions for several states does not preserve consistency
- Some actions in a action layer are not independent
- How to capture incompatibility between actions and propositions ?

Solution

The solution is to keep track incompatible actions and propositions also called mutual exclusion relations based on action independent criteria

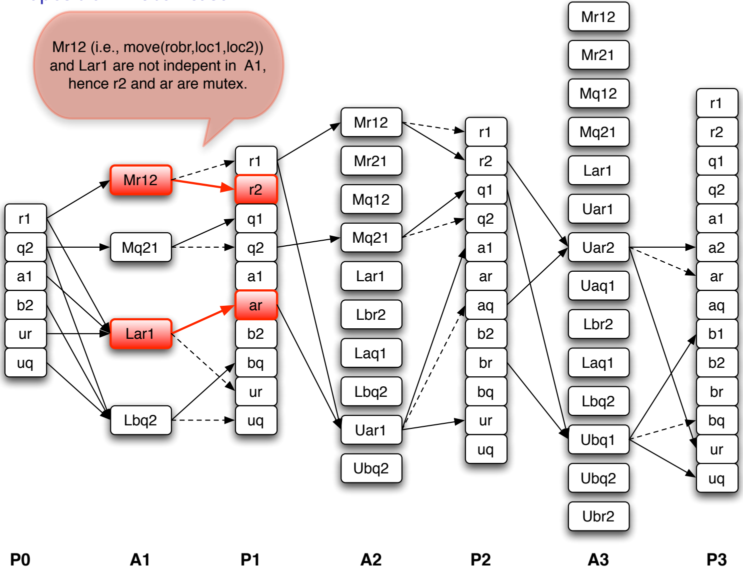
Mutual Exclusion Relations

Proposition Mutex

1. Two dependent actions in an action layer cannot appear simultaneously, hence the positive effects of two dependent actions are incompatible unless these propositions are also positive effects of some other independent actions
   - Two propositions are incompatible in the sens where they cannot be reached through a single level
2. Negative and positive effects of an action are also incompatible propositions
   - to deal with this second type of incompatibility, it is convenient to introduce for each proposition p a dummy action called no-op, noted α_p, whose precondition and sole effect is p
   - if an action a has p as a negative effect, then according to our definition, a and α_p are independent actions (positive effects incompatible)

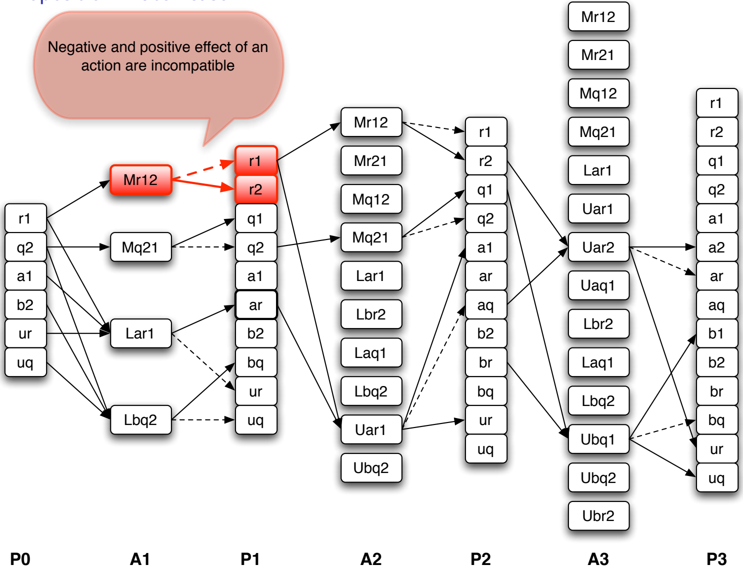
Mutual Exclusion Relations

Example: Proposition Mutex case 1



Mutual Exclusion Relations

Example: Proposition Mutex case 2



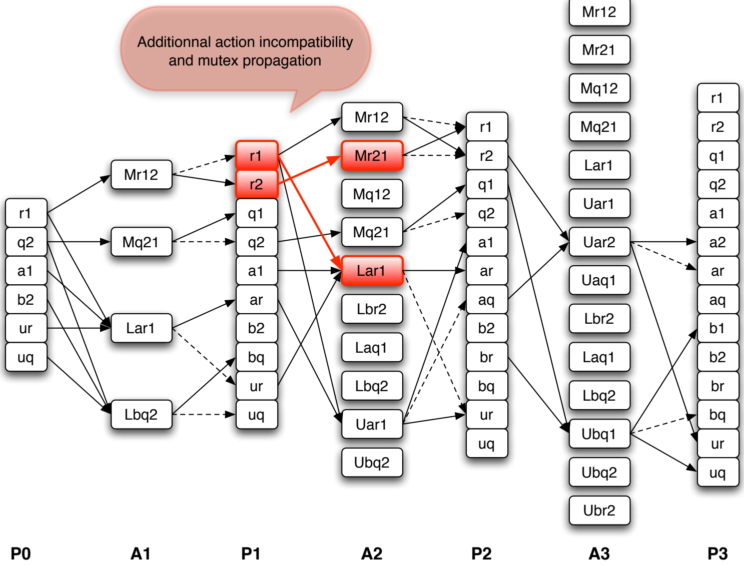
Mutual Exclusion Relations

Action Mutex and Mutex Propagation

- Dependency between actions in an action level A_i of the planning graph leads to incompatible proposition in a level P_i
- Conversely, incompatible propositions in a level p_i lead to additionnal incompatible actions in the following level A_{i+1}
- These are actions whose preconditions are incompatible

Mutual Exclusion Relations

Example: Action Mutex and Mutex Propagation



Mutual Exclusion Relations

Definition

Definition (Mutual Exclusion Relation)

- Two actions a and b in level A_i are mutex if :
  1. a and b are dependent or
  2. a precondition of a is mutex with a precondition of b
- Two propositions p and q in P_i are mutex if:
  1. every action in A_i that has p as positive effect (including no-op actions) is mutex with every action that produces q and
  2. there is no action in A_i that produces both p and q

Note

- Dependent actions are necessarily mutex
- Dependency is an intrinsic property of the actions in a domain, while the mutex relation takes into account additionnal constraints of the problem
- For a same problem, a paire of actions may be mutex in some action level A_i and become non-mutex in some latter level A_j of a planning graph

Mutual Exclusion Relations

Example[1]

Level	Mutex elements
A_1	$\{Mr12\} \times \{Lar1\}$
	$\{Mq21\} \times \{Lbq2\}$
P_1	$\{r_2\} \times \{r_1, a_r\}$
	$\{q_1\} \times \{q_2, b_r\}$
	$\{a_r\} \times \{a_1, u_r\}$
	$\{b_q\} \times \{b_2, u_q\}$
A_2	$\{Mr12\} \times \{Mr21, Lar1, Uar1\}$
	$\{Mr21\} \times \{Lbr2, Lar1^*, Uar1^*\}$
	$\{Mq12\} \times \{Mq21, Laq1, Lbq2^*, Ubq2^*\}$
	$\{Mq21\} \times \{Lbq2, Ubq2\}$
	$\{Lar1\} \times \{Uar1, laq1, Lbr2\}$
	$\{Lbr2\} \times \{Ubq2, Lbq2, Uar1, Mr12^*\}$
	$\{Laq1\} \times \{Uar1, Ubq2, Lbq2, Mq21^*\}$
	$\{Lbq2\} \times \{Ubq2\}$
P_2	$\{b_r\} \times \{r_1, b_2, u_r, b_q, a_r\}$
...	...

[1] A star (*) denotes mutex actions that are independent but have mutex preconditions

Mutual Exclusion Relations

Notation

- We note the set of mutex pairs in A_i as μA_i and the set of mutex pairs in P_i as μP_i
- Let us remark that:
    1. dependency between actions as well as mutex between actions or propositions are symmetrically relations
    2. for $\forall i : P_{i-1} \subseteq P_i$ and $A_{i-1} \subseteq A_i$

Mutual Exclusion Relations

Planning Graph Monotonicity

Proposition (Monotonicity)

If two proppropositions p and q are in P_{i-1} and $(p, q) \notin \mu P_{i-1}$, then $(p, q) \notin \mu P_i$ and if two actions a and b are in A_{i-1} and $(a, b) \notin \mu A_{i-1}$, then $(a, b) \notin \mu A_i$.

Proof

Every proposition p in a level P_i is supported by at least its no-op action α_p. Two no-op actions are necessarily independent. If p and q in P_{i-1} are such that $(p, q) \notin \mu P_{i-1}$, then $(\alpha_p, \alpha_q) \notin \mu A_i$. Hence, a non-mutex pair of propositions remains non-montex in the following level. Similarly, if $(a, b) \notin \mu A_{i-1}$, then a and b are independent and their preconditions in P_{i-1} are not mutex; both properties remain valid at the following level.

Mutual Exclusion Relations

Planning Graph Monotonicity

- According to this result,
  - propositions and actions in a planning graph monotonically increase from one level to the next
  - mutex pairs monotonically decrease
- These monotonicity properties are essential to the complexity and the terminaison of the planning graph techniques

Proposition (Weak Reachability)

A set g of propositions is reachable from s_0 only if:

- there is in the corresponding planning graph a proposition layer P_i such that $g \in P_i$ and
- no pair of propositions in q are in μP_i

The Graphplan Planner

- The Graphplan algorithm performs a procedure close to interative deepening, discovering a new part of the search space at each iteration. It iteratively:
  1. expands the planning graph by one level and
  2. searches backward form the last level of this graph for a solution
- The fisrt extraction, proceeds to level P_i in which all of the goal propositions are included and no paires of them are mutex
  - it does not make sens to start searching a graph that does not meet the necessary condition of the weak reachability
- The iterative loop of graph expansion and search is pursued until either a plan is found or a failure terminaison condition is met

Expanding the Planning Graph

Planning Graph

- Let (O, s_0, g) be a planning problem in the classical representation such that s_0 and g are set of propositions, and operators in O have no negated literals in their preconditions
- Let A be the union of all ground instances of operators in O and of all no-op actions α_p for every proposition p of that problem
  - the no-op action for p is defined as
    - precond(α_p) = effects$^+(\alpha_p)$, and
    - effects$^-(\alpha_p) = \emptyset$
- A planning graph for a planning problem expanded up to level i is a sequence of layers of nodes and of mutex pairs:

$$G = \langle P_0, A_1, \mu A_1, P_1, \mu P_1, \ldots, A_i, \mu A_i, P_i, \mu P_i \rangle$$

Expanding the Planning Graph

Procedure

- The planning graph does not depend on g
  - it can be used for different planning problem that have the same set of planning operators O and initial state s_0
- The expansion of G starts initially from $P_0 \rightarrow s_0$
- The expansion procedure correspond to generate the set A_i, P_i, μA_i and μP_i, respectively from the elements in the previous level $i - 1$

Expanding the Planning Graph

Procedure

Algorithm (Expand($\langle P_0, A_1, \mu A_1, \ldots, A_{i-1}, \mu A_{i-1}, P_{i-1}, \mu P_{i-1} \rangle$))

$A_i \leftarrow \{a \in A \mid precond(a) \in P_{i-1}$ *and* $precond(a) \cap \mu P_{i-1} = \emptyset\}$
$P_i \leftarrow \{p \mid \exists a \in A_i : p \in effects^+(a)\}$
$\mu A_i \leftarrow \{(a, b) \in A_i, a \neq b \mid a, b$ *are dependent*
  or $\exists (p, q) \in \mu P_{i-1} : p \in precond(a)$ *and* $q \in precond(b)$ $\}$
$\mu P_i \leftarrow \{(p, q) \in P_i, p \neq q \mid \forall a, b \in A_i, a \neq b :$
  $p \in effects^+(a)$ *and* $q \in effects^+(b) \Rightarrow (a, b) \in \mu A_i\}$
foreach $a \in A_i$ **do**
  link a *with a precondition arcs to* $precond(a)$ *in* P_{i-1}
  link a *with a positive arcs to* $effects^+(a)$ *in* P_i
  link a *with a negative arcs to* $effects^-(a)$ *in* P_i
end
return $\langle P_0, A_1, \mu A_1, \ldots, P_{i-1}, \mu P_{i-1}, A_i, \mu A_i, P_i, \mu P_i \rangle$

Expanding the Planning Graph

Complexity

Proposition

The size of a planning graph down to level k and the time required to expand it to that level are polynomial in the size of the planning problem.

Proof

If the planning problem (O, s_0, g) has a total of n propositions and m actions, then $\forall i : |P_i| \leq n$, and $|A_i| \leq m + n$ (including no-op actions), $\mu A_i \leq (m+n)^2$, and $|\mu P_i| \leq n^2$. The steps involved in the generation of these set are of polynomial complexity in the size of the sets.
Furthermore, n and m are polynomial in the size of the problem (O, s_0, g). This is the case because, according to classical planning assumptions, operators cannot create new constant symbols. Hence, if c is the number of constant symbols given in the problem, $e = max_{o \in O}\{|\text{effects}^+(o)\}$, and α is an upper bound on the number of parameters of any operators, then $m \leq |O| \times c^\alpha$, and $n \leq |s_0| + e \times |O| \times c^\alpha$.

Expanding the Planning Graph

Planning Graph Fixed-point

- The number of distinct levels in a planning graph is bounded
- At some stage, the graph reached a fixed-point

Definition (Fixed-point Level)

A fixed-point level in a planning graph G is a level κ such that for $\forall i, i > \kappa$, level i of G is identical to level κ, 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-poiny level κ, which is the smallest k such that $|P_{k-1}| = |P_k|$ and $|\mu P_{k-1}| = |\mu P_k|$.

Proof

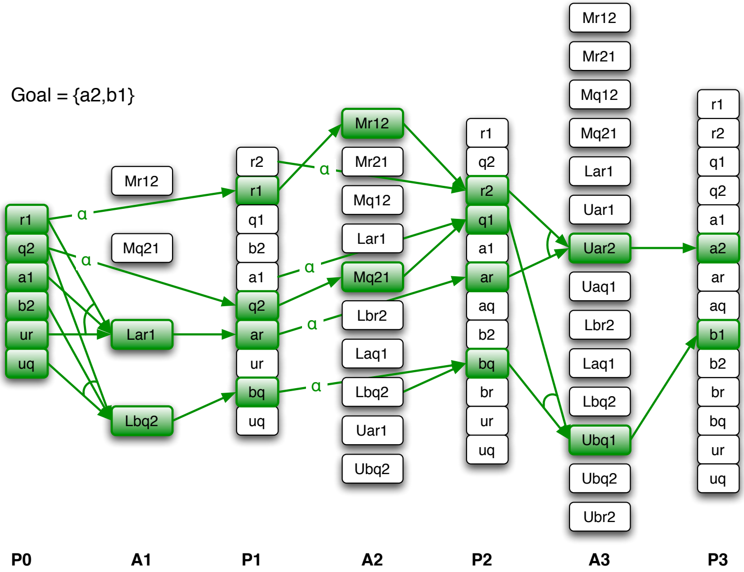
To do ...

Searching the Planning Graph

- The search for a solution plan in a planning graph proceeds back from a level P_i that includes all goal proposition, no pair of which is mutex, i.e., $g \in P_i$ and $g \cap \mu P_i = \emptyset$.
- The search procedure looks for a set $\pi \in A_i$ of non-mutex actions that achieve these propositions.
- Preconditions of elements of π becomes the new goal for levem $i - 1$ and so on
- A failure to meet the goal of some level j leads to a backward over other subsets of A_{j+1}
- If level 0 is successfully reached, then the corresponding sequence $\langle \pi_1, \ldots, \pi_i \rangle$ is a solution plan

Searching the Planning Graph

Example



Searching the Planning Graph

Remark

- The extraction of a plan from a planning graph corresponds to a search in an AND/OR subgraph of the planning graph:
  - From a proposition in goal g, OR-branches are arcs from all actions in the preceding action level that support this proposition, i.e., positive arcs to that proposition
  - From an action node, AND-branches are its preconditions arcs

Searching the Planning Graph

No-goods

- The mutex relation between propositions provides only forbidden pairs, not tuples
- The search may show that a tuple or more that two propositions corresponding to an intermediate subgoal fails
  - Because of the backtracking and iterative deepening, the search may have to analyse that same tuple more than once
- Recording tuples that failed may save time in future searched
  - This recording is performed into a no-good hash-table denoted ∇
  - This hash-table is indexed by the level of the fail goal because goal g may fail at level i and succeed at $j > i$

Searching the Planning Graph

Extract Procedure

- The extract procedure takes as input:
  - a planning graph G
  - a current set of goal propositions g and
  - a level i
- The procedure extracts a set of actions $\pi \subseteq A_i$ that achieves propositions of g by recursively a other procedure that try to establish g at level i
- If the procedure succeeds in reaching level 0, then it returns an empty sequence, from which pending recursions successfully return a solution plan
- It records failes tuples into ∇ table, and it check each current goal with respect to recorded tuples
  - Note: a tuple g is added to the no-good table at level i only if the call to establish g at level i fails

Searching the Planning Graph

Extract Procedure

Algorithm (Extract(G, g, i))

if $i = 0$ **then return** $\langle\rangle$
if $g \in \nabla(i)$ **then return failure**
$\pi \leftarrow \texttt{GP-Search}(G, g, \emptyset, i)$
if $\pi \neq$ **failure then return** π
$\nabla(i) \leftarrow \nabla(i) \cup \{g\}$
return failure

Searching the Planning Graph

GP-Search Procedure

- The GP-Search procedure selects each goal proposition p at a time, in some heuristic order
- Then, it nondeterministically chooses among the the resolvers of p one action a that tentatively extends the current subset π
  - The resolvers are actions that achieve p and that are not mutex with action already selected at that level
- Then it recursively calls the same procedure
  - The recursive call is done on a subset of goals minus p and minus all positive effect of a in g
  - A failure for this non-deterministic choice is a backtracking further up if all resolvers of p have been tried
- When g is empty, then π is complete and the search recursively tries to extract a solution for the folowing level $i - 1$

Searching the Planning Graph

GP-Search Procedure

Algorithm (GP-Search(G, g, π, i))

if $g = \emptyset$ **then**
    $\Pi \leftarrow$ `Extract`$(G, \bigcup\{precond(a) \mid \forall a \in \pi\}, i-1)$
    if Π = **failure then return failure**
    return $\Pi.\langle\pi\rangle$
else
    select any $p \in g$
    resolvers $\leftarrow \{a \in A_i \mid p \in effects^+(a) \text{ and } \forall b \in \pi : (a, b) \notin \mu A_i\}$
    if resolvers $= \emptyset$ **then return failure**
    nondeterministically choose $a \in$ resolvers
    return `GP-Search`$(G, g - effects^+(a), \pi \cup \{a\}, i)$
end

Searching the Planning Graph

Graphplan Procedure

- Graphplan performs an initial graph expansion until
  1. it reaches a level containing all goal propositions without mutex or
  2. it arrives at a fixed-point level in G
- If condition 2 happens first, then the goal is not achievable
- Otherwise, a search for a solution is performed and if no solution is found at this stage, the algorithm iteratively expands and then searches the graph G
- This iteractive deepening is pursued even after a fixed-point level has been reached until
  1. success or
  2. the terminaison condition is satisfied
     - ★ This terminaison condition requires that the number of no-goods tuples in $\nabla(\kappa)$ at the fixed-point level κ, stabilizes after two successive failures

Searching the Planning Graph

Graphplan Procedure

Algorithm (Graphplan(A, s_0, g))

$i \leftarrow 0,\ \nabla \leftarrow \emptyset,\ P_0 \leftarrow s_0,\ G \leftarrow \langle P_0 \rangle$
repeat
    $i \leftarrow i+1,\ G \leftarrow \texttt{Expand}(G, g, i)$
until $[g \subseteq P_i \text{ or } g \cap \mu P_i = \emptyset]$ *or* $\texttt{Fixedpoint}(G)$
if $g \not\subseteq P_i$ *or* $g \cap P_i \neq \emptyset$ **then return failure**
$\Pi \leftarrow \texttt{Extract}(G, g, i)$
if $\texttt{Fixedpoint}(G)$ **then** $\eta \leftarrow |\nabla(\kappa)|$ **else** $\eta \leftarrow 0$
while $\Pi =$ **failure do**
    $i \leftarrow i+1,\ G \leftarrow \texttt{Expand}(G, g, i),\ \Pi \leftarrow \texttt{Extract}(G, g, i)$
    if $\Pi =$ **failure** *and* $\texttt{Fixedpoint}(G)$ **then**
        if $\eta = |\nabla(\kappa)|$ **then return failure**
    end
    $\eta \leftarrow |\nabla(\kappa)|$
end
return Π

Analysis of Graphplan

Soundness, Completeness, and Terminaison

- We must analyse how the no-goods table evolves along successive deepening stages of G
- Let $\nabla_j(i)$ be the set of no-good tuples found at level i after the successful completion of a deepening state down to a level $j > i$
- The failure of stage j means that
  - any plan j or fewer steps must make at least one of the goal tuples in $\nabla_j(i)$ true at level i and
  - none of these tuples is achievable in i levels

Proposition

$\forall i, j$ such that $j > i, \nabla_j(i) \subseteq \nabla_{j+1}(i)$

Analysis of Graphplan

Soundness, Completeness, and Terminaison

Proof

A tuple of goal proposition g is added as a no-good in $\nabla_j(i)$ only when Graphplan has performed an exhaustive search for all ways to achieve g with the actions in A_i and tit fails : each set of actions in A_i that provides g is either mutex or involves a tuple of preconditions g' that was shown to be a no-good at the previous level $\nabla_k(i-1)$, for $i < k \leq j$. In other words, only the levels from 0 to i in G are responsible for the failure of the tuple g at level i. By iterative deepening, the algorithm may find that g is solvable at some level $i' > i$, but regardless of how many iterative deepening stages are performed, once g is in $\nabla_j(i)$, it remains in $\nabla_{j+1}(i)$ and in the no-good table at level i in all subsequent deepening stages.

Analysis of Graphplan

Soundness, Completeness, and Terminaison

Proposition

The Graphplan algorithm is sound and complete, and it terminates. It returns *failure* iff the planning problem (O, s_0, g) has no solution; otherwise, it returns a sequence of sets of actions Π that is a solution plan to the problem.

Proof

To do . . . (use previous proposition)

Analysis of Graphplan

Remarks

1. The mutex relation on incompatible pairs of actions and propositions, and weak reachability condition, offer a very good insight about the interaction between the goals of a problem and about which goals are possibly achievable at some level
2. Because of the monotonic properties of the planning graph, the algorithm is guaranteed to terminate
3. The fixe-point feature together with reachability condition provide an efficient failure terminaison condition
   - In particular, when the goal propositions without mutex are not reachable, no search at all is performed

Analysis of Graphplan

Conclusion

- Because of its backward constraint-directed search, Graphplan bought a significant speed-up and contributed to the scalability of planning
- Evidently, Graphplan does not change the intrinsic complexity of planning, which is PSPACE-complete in the set-theorie representation
- Since the expansion of the planning graph is in polynomial time, this means that the costly part of the algorithm is in the search of the planning graph
- The memory requirement of the planning graph data structure can be a significant limiting factor
- Severals techniques and heuristics have been devised to speed-up the search and to improve the memory management of its data structure

Extanding the Language

Handling negation

- Handling negation in the preconditions of operators and in goals is easilu performed by introducing a new predicate not-op to replace the negation of a predicate p in precondition or goal
- This replacement requires
  1. adding not-p in effects$^-$ when p is in effects$^+$ of an opperator o and
  2. adding not-p in effects$^+$ when p is in effects$^-$ of o
- One also has to extend s_0 with respect to newly introduced not-p predicate in order to maintain a consistent and closed initial world
  - That is, any proposition that is not explicitly stated is false

Extanding the Language

Handling negation example

Example

The DWR domain has the following operator:

move(r, l, m) *;; robot r at location l moves at a connected location m*

precond: at(r, l), adjacent(l, m), ¬occupied(m)
effects: at(r,m), occupied(m), ¬occupied(l), ¬at(r, l)

The negation in the precondition is handled by introducing the predicate not-occupied in the following way:

move(r, l, m) *;; robot r at location l moves at a connected location m*

precond: at(r, l), adjacent(l, m), not-occupied(m)
effects: at(r,m), occupied(m), ¬occupied(l), ¬at(r, l), not-occupied(l), ¬not-occupied(m)

Furthermore, if a problem has three locations (l1, l2, l3) such that only l1 is initially occupied, we need to add to the initial state the propositions:

- not-occupied(l2)
- not-occupied(l3)

Extanding the Language

Remarks

- This approach, which rewrites a planning problem into restricted representation required by Graphplan, can also be used to handling other extensions.
- For example, recall that an operator with a conditional effect can be expanded into set of pairs (precond$_i$, effects$_i$).
  - Hence it is easy to rewrite it as several operators, one for each such pair
  - Quantified conditionan effects are similary expanded
- Such expansion may lead to an expontial number of operators
  - It is preferable to generalize the algorithm for directly handling an extended language

Extanding the Language

Generalizing Graphplan with disjunctive preconditions

- Generalizing Graphplan for directly handling operators with disjunctive preconditions can be done by considering the edges from an action in A_i to its preconditions in P_{i-1} as being a disjunctive set of AND-connectors, as in AND/OR graph
- The definition of mutex between actions needs to be generalized with respect to these connectors
- The set of resolvers in GP-Search, among which a nondeterministic choice is made for achieving a goal, now has to take into account not the actions but their AND-connector

Extanding the Language

Generalizing Graphplan with conditional effects

- Directly handling operators with conditional effects requires more significant modifications
- One has to start with generalized definition of dependency between actions taking into accouny their conditional effects
  - This is needed in order to keep the desirable result of proposition on applicable actions' set (an independent set of actions defines the same state transitions for any permutation of the set)
- One also has to define a new structure of planning graph for handling the conditional effects
  - For example, for propagating a desired goal at level P_i, which is a conditional effect over to its antecedent condition either in a positive or in a negative way
- One also has to come up with ways to compute and propagate mutex relations and with a generalization of the search procedure in the new planning graph (cf. IPP Planner)

Improving the Planner

Memory Management

- The planning graph data structure makes explicit all the ground atoms and instantiated actions of a problem
  - it has to be implemented carefully in order to maintain reasonable memory demand that is not a limiting factor on the planner's performance
- The monotonic properties of the planning graph are essential to this purpose.
  - Because $P_{i-1} \subseteq P_i$ and $A_{i-1} \subseteq A_i$, one does not need to keep these sets explicitly but record for each proposition p the level i at wich p appeared for the first time in the graph, and similarly for each action
  - A symetrically technique can be used for mutex relation
- There is no need to record the graph after its fixed-point κ
- Finally, several general programming techniques can be useful for memory management
  - For example, the bitvector data structure allows one to encode a state and a proposition level P_i as a vector of n bits, where n is the number of propositions in the problem and actions is encoded as four such vectors, one for each of its positive and negative preconditions and effects

Improving the Planner

Focusing and Improving the Search: Removing Rigid Predocates

- The description of a domain involves rigid predicates that does not vary from state to state
  - In the DWR domain, the predicate adjacent, attached and belong are rigid
  - There is no operator that changes their truth values
- Once operators are instanciated into ground actions for a given problem, one may remove the rigid predicates from preconditions and effects because they play no further role in the planning process
- This simplication reduce the number of actions
- This preprocessing can be quite sophisticated and may allow one to infer nonobvious types, symetries, and invariant properties, such as permanent mutex relations.

Improving the Planner

Focusing and Improving the Search: The No-good table

- No-good tuples, as well as mutex relations play an essential role in pruning the search
- Howerver, if we are searching to achieve a set of goals g in level i, and if $g' \in \nabla_i$ such that $g' \subset g$, we will not detect that g is not achievable and prune the search
- The Extract procedure can be extended to test this type of set inclusion
  - But, this may involve a significant overhead (cf. UBTree structure for a efficient test of inclusion)
- The problem of this improvement consists of turning out the terminaison condition of the algorithm $(|\nabla_{j-1}(i)| = |\nabla_j(\kappa)|)$ holds even if the procedure records and keeps in ∇_i only no-good tuples g such that no subset of g has been proven to be a no-good

Improving the Planner

Focusing and Improving the Search: Heuristics

- GP-Search procedure has to be focused with heuristics for:
  1. selecting the next proposition p in the current set g
  2. nondeterministically choosing the action in the resolvers
- A general heuristics consists of selecting first a proposition p that lead to the smallest set of resolvers, i.e., the propositions p achieved by the smallest number of actions
- A symetrically heuristics for the choice of an action supporting p is to prefer no-op action first
- Other heuristics that are more specific to the planning graph structure and more informed take into account the level at which actions and propositions appear for the first time in the graph
  - The later a proposition appears in the planning graph, the most constrainted it is
  - Hence, one would select the latest proposition first

Improving the Planner

Focusing and Improving the Search: CSP Techniques

- A number of algorithmic techniques allow one to improve the efficiency of the search
- For example, one is the forward-checking technique:
  - Before choosing an action a in *resolvers* for handling p, one checks that this choice will not leave another pending proposition in g with an empty set of resolvers.
- Forward-checking is a general algorithm for solving constraint satisfaction problem

Extending the Independence Relation

- The concept of layered plans is defined with a strong requirement of independent actions in each set π
- In practice, we do not necessarily need to have every permutation of each set be a valid sequence of actions
- We only need to ensure that there is at least one such permutation
- This is the purpose of the relation between action called allowance relation, which is less constrained than the independence relation while keeping the advantages of the planning graph

Extending the Independence Relation

Allowance Relation

- An action a allows an action b when b can be applied after a and the resulting state contains the union of the positive effects of a and b
- This is the case when a does not delete a precondition of b and b does not delete a positive effect of a:
  - a allows b iff $\text{effects}^-(a) \cap \text{precond}(b) = \emptyset$ and
  - $\text{effects}^-(b) \cap \text{effects}^+(a) = \emptyset$
- Allowance is weaker than independence
- Independence implies allowance:
  - If a and b are independent, then a allows b and b allows a
  - Note that when a allows b but b does not allow a, then a has to be ordered before b
  - Note also that allowance is not symmetrical relation

Extending the Independence Relation

Allowance Relation and Mutex Relation

- If we replace independence relation with allowance relation in the mutex definition, we can say that two actions *a* and *b* are mutex either:
  1. when they have mutually exclusive preconditions, or
  2. when *a* does not allow *b* and *b* does not allow *a*
- This definition leads to fewer mutex pairs between actions, and consequently to fewer mutex relation between propositions
- On the same planning problem, the planning graph will have fewer or at most the same number of levels, before reaching a goal or fixed-pointn than with the independence relation

Extending the Independence Relation

Example : Allowance Relation

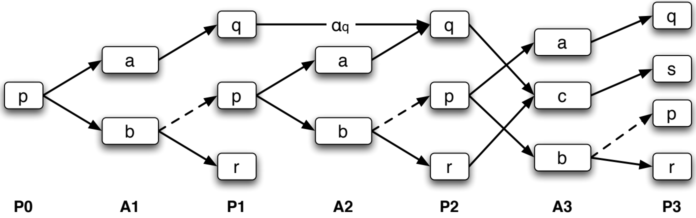
Example

- Let a simple planning domain that has three actions (a, b and c) and four propositions (p, q, r and s):
  - $\text{precond}(a) = \{p\}$; $\text{effects}^+(a) = \{q\}$; $\text{effects}^-(a) = \{\}$
  - $\text{precond}(b) = \{p\}$; $\text{effects}^+(b) = \{r\}$; $\text{effects}^-(b) = \{p\}$
  - $\text{precond}(c) = \{q, r\}$; $\text{effects}^+(c) = \{s\}$; $\text{effects}^-(c) = \{\}$
- Action a and b are not independent (b deletes a precondition of a
  - Hence, they will be mutex in any level of the planning graph
- Action a allows b:
  - these actions will not be mutex with the allowance relation

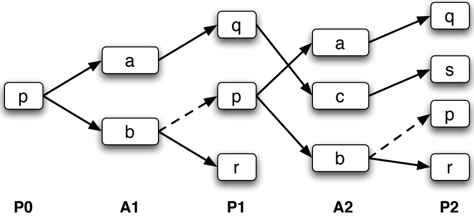
Extending the Independence Relation

Example: Allowance Relation

Graph with independence relation



Graph with allowance relation



Extending the Independence Relation

- The benefit of the allowance relation (fewer mutex pair and a smaller fixed-point) has a cost
- Since allowance relation is not symetrical, a set of pairwise nonmutex actions does not necessarily contain a "valid" permutation
  - For instance, if a allows b, b allows c and c allows a but none of the opposite relations holds, then the three actions a, b and c can be nonmutex.
  - But there is no permutation that gives an applicable sequence of actions and a resulting state corresponding to the union of their positive effects
  - Remember that earlier a set of nonmutex actions was necessarily independent and could not be selected in the search phase for a plan (here we have to add a further requirements for the allowance relation within a set)

Extending the Independence Relation

- A permutation $\langle a_1, \ldots, a_n \rangle$ of the elements of a set π is allowed if every action allows all its followers in the permutation:

$$\forall i, k : \text{if} j < k, \text{then} a_j \text{allows} a_k$$

- A set is allowed if it has at least one allowed permutation
- The state resulting from the application of an allowed set can be defined as previously:

$$\gamma(s, \pi_i) = (s - \text{effects}^-(\pi_i)) \cup \text{effects}^+(\pi)$$

- All previous propositions remain valid

Extending the Independence Relation

- In order to compute $\gamma(s, \pi_i)$ and to use such as a set in the GP-Search procedure one does not need to produce an allowed permutation and to commit the plan to it one just needs to check its existence
- We already noriced that an ordering constraints "*a* before *b*" would be required whenever *a* allows *b* but *b* soies not allows *a*
- It is easy to prove that a set is allowed if the relation consisting of all pairs (a, b) such that "*b* doies allow *a*" is cycle free.
- This can be checked with a topological sorting algorithm in complexity that is linear in the number of actions and allowance pair
- Such a test must be take place in the GP-Search procedure

Extending the Independence Relation

Modifying GP-Search procedure

Algorithm (GP-Search(G, g, π, i))

if $g = \emptyset$ **then**
    if π_i *is not allowed* **then** **return failure**
    $\Pi \leftarrow \texttt{Extract}(G, \bigcup\{precond(a) \mid \forall a \in \pi\}, i-1)$
    if $\Pi =$ **failure then return failure**
    return $\Pi.\langle\pi\rangle$
else
    select any $p \in g$
    resolvers $\leftarrow \{a \in A_i \mid p \in effects^+(a) \text{ and } \forall b \in \pi : (a,b) \notin \mu A_i\}$
    if resolvers $= \emptyset$ **then return failure**
    nondeterministically choose $a \in$ resolvers
    return $\texttt{GP-Search}(G, g - effects^+(a), \pi \cup \{a\}, i)$
end

Extending the Independence Relation

Conclusion

- The modifications bring to Graphplan to take into account allowance relation keep Graphplan sound and complete
- The allowance relation lead to fewer mutex pairs, hence to more action in a level and to fexer level in the planning graph
- The reduced search space increase the performance of the algorithm
- The benefit can be very significant for highly constraint problem where the search phase is very expensive

Exercices

Exercice 1

Let $P = (O, s_0, g)$ and $P' = (O, s_0, g')$ be the statements of two solvable planning problems such that $g \subseteq g'$. Suppose we run Graphplan on both problems, generating planning graphs G and G'. Is $G \subseteq G'$?

Exercice 2

Detail the modifications required for handling operator with disjunctive preconditions in the modification if mutex relations and in the planning procedures.

Exercice 3

Discuss the strucrure of plans as output by Graphplan with allowance relation. Compare these plans to sequences of independent sets of actions, to plan that are simple sequences of actions, and to partially ordered sets of actions.

Further readings

A.Blum and M.Furst.
Fast planning through planning graph analysis
Artificial Intelligence, 90(1-2):281 300, 1997

Kambhampati, E.Parker, and E.Lambrecht.
Understanding and extending Graphplan
In Proceedings of the European Conference on Artificial Intelligence, pages 260- 272, 1997

A.Gerevini and I.Serina.
A planner based on local search for planning graphs with action costs
In Proceedings of the Artificial Intelligence Planning Systems, pages 13 -22, 2002.

J. Koehler.
Handling of conditional effects and negative goals in IPP
Technical report, Freiburg University, 1999.