

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
 - ▶ \implies Planning-Graph is less expressive than Plan-space but more than State-space
- Planning-Graph approach rely on two interrelated ideas:
 - 1 **Reachability analysis**: addresses the issue of whether 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

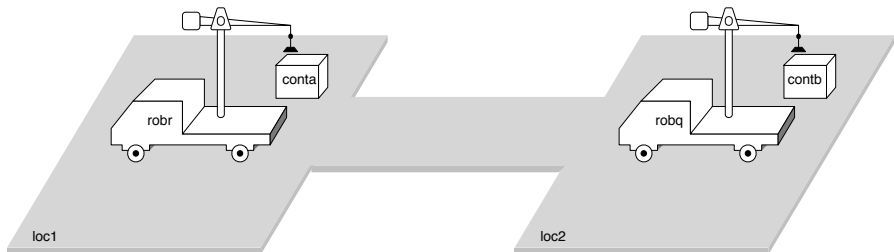
Definition (Reachability)

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



Example: Reachability Trees

Operators

$\text{move}(r, l, l')$;; robot r at location l moves at a connected location l'

precond: $\text{at}(r, l), \text{adjacent}(l, l')$

effects: $\text{at}(r, l'), \neg \text{at}(r, l)$

$\text{load}(c, r, l)$;; robot r at location l loads container c

precond: $\text{at}(r, l), \text{in}(c, l), \text{unloaded}(r)$

effects: $\text{loaded}(r, c), \neg \text{in}(c, l), \neg \text{unloaded}(r)$

$\text{unload}(c, r, l)$;; robot r at location l unloads container c

precond: $\text{at}(r, l), \text{loaded}(r, c)$

effects: $\text{unloaded}(r), \text{in}(c, l), \neg \text{loaded}(r, c)$

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

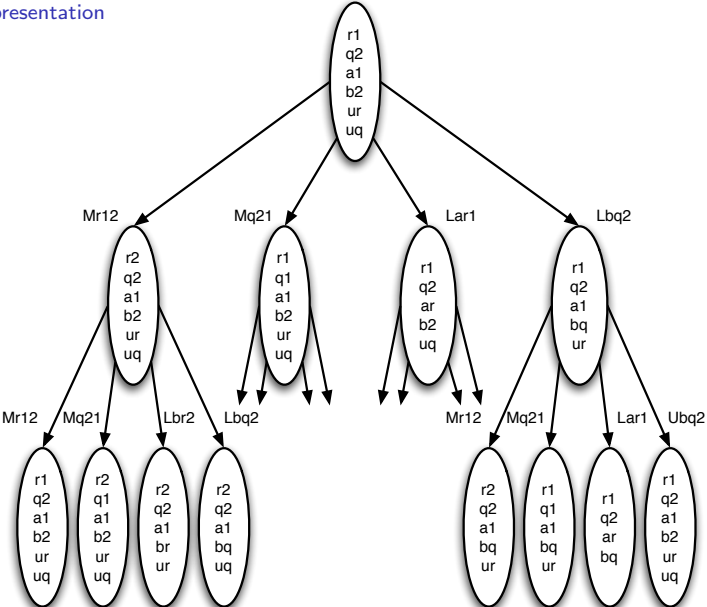
Example: Reachability Trees

Notation

- To simplify representation, let us denote atoms by propositional symbols:
 - ▶ r_1 and r_2 stand for $\text{at}(\text{robr}, \text{loc1})$ and $\text{at}(\text{robr}, \text{loc2})$
 - ▶ q_1 and q_2 stand for $\text{at}(\text{robq}, \text{loc1})$ and $\text{at}(\text{robq}, \text{loc2})$
 - ▶ a_1 , a_2 , a_r and a_q stand for $\text{int}(\text{conta}, \text{loc1})$, $\text{in}(\text{conta}, \text{loc2})$, $\text{loaded}(\text{conta}, \text{robr})$ and $\text{loaded}(\text{conta}, \text{robq})$
 - ▶ b_1 , b_2 , b_r and b_q stand for $\text{int}(\text{contb}, \text{loc1})$, $\text{in}(\text{contb}, \text{loc2})$, $\text{loaded}(\text{contb}, \text{robr})$ and $\text{loaded}(\text{contb}, \text{robq})$
- Let us also denote the 20 actions in A :
 - ▶ Mr_{12} is the action $\text{move}(\text{robr}, \text{loc1}, \text{loc2})$, Mr_{21} is the opposite, and Mq_{12} and Mq_{21} are the similar move action for robot robq
 - ▶ Lar_1 is the action $\text{load}(\text{conta}, \text{robr}, \text{loc1})$ Lar_2 , Laq_1 and Laq_2 are the other load actions for conta in loc2 with contb. Lbr_1 , Lbr_2 , Lbq_1 and Lbq_2 are the load actions for contb
 - ▶ Uar_1 , Uar_2 , Uaq_1 , Uaq_2 , Ubr_1 , Ubr_2 , Ubq_1 , and Ubq_2 are the unload actions

Example: Reachability Trees

Graphic representation

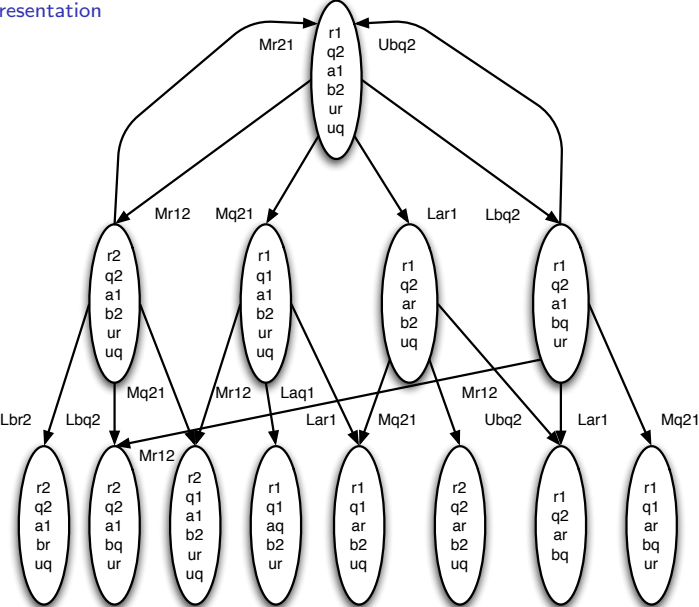


Reachability Trees

- 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
⇒ 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 several states

Reachability with Planning Graphs

Basic Idea

Reachability tree	Planning Graph
<ul style="list-style-type: none">• Actions branching out from a node are mutually exclusive• A node is associated with the proposition that necessarily hold for that node• State is a consistent set of propositions	<ul style="list-style-type: none">• Actions are considered as inclusive disjunction from a node to the next that contains all the effects of the actions• A node contains proposition that possibly hold at some point• The union of the sets of propositions for several states does not preserve consistency \implies solution is to keep track incompatible actions and propositions

Reachability with Planning Graphs

Informal Definition

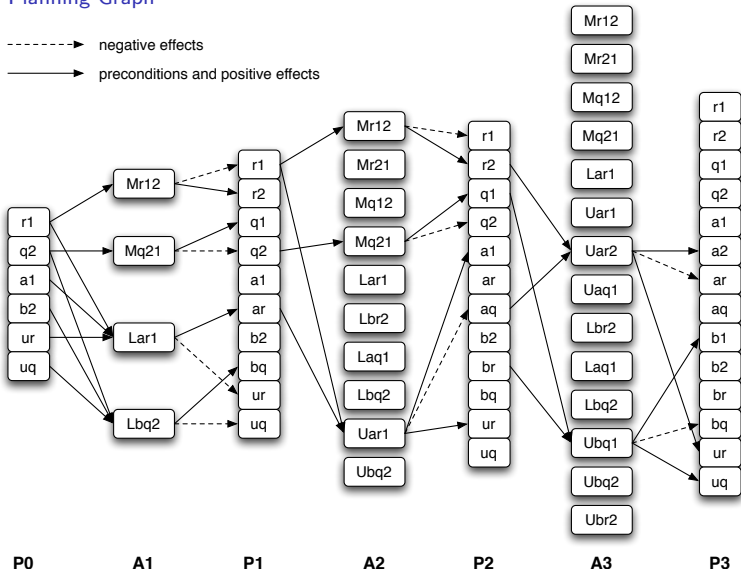
- A planning graph is a **directed layered graph**:
 - ▶ arcs are permitted only from one layer to the next
- Nodes in level 0 correspond to the set P_0 of propositions denoting the initial state s_0 of the planning problem
- Level 1 contains two layers:
 - 1 an action level A_1 that is the set of actions (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

Example: Planning Graph

-----> negative effects

————> preconditions and positive effects



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**

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

- ▶ A plan is qualified as **layered plan** since it 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 independent 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

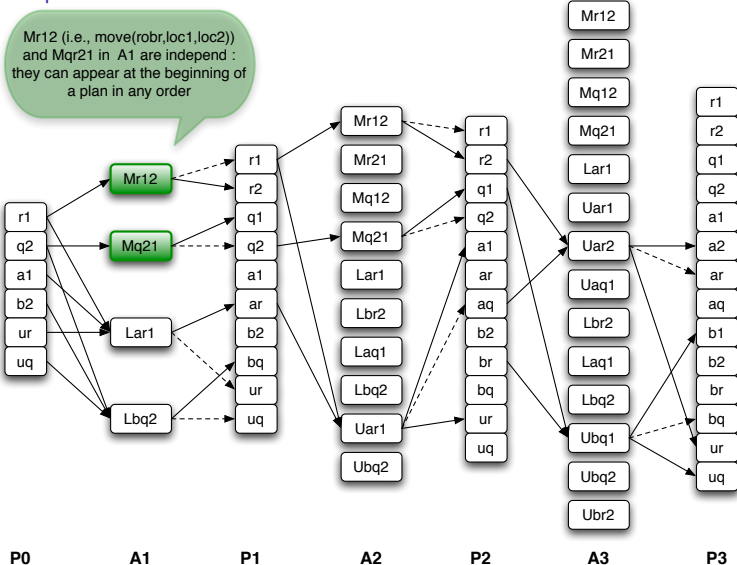
A set of actions π is independent when every pair of π is independent

- 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
 - ▶ symmetrically for negative effects of b with respect to a
 - ★ b deletes a precondition on a positive effect of a

Independent Actions

Example: independent actions

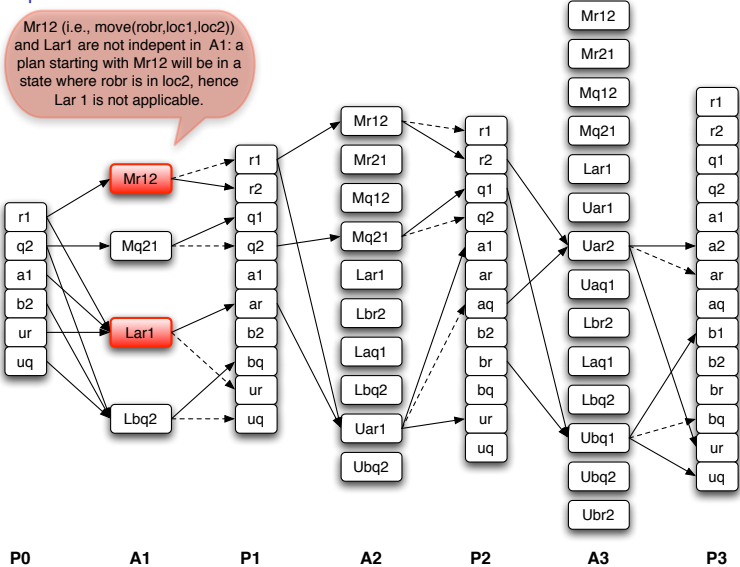
Mr12 (i.e., move(robr,loc1,loc2)) and Mqr21 in A1 are independent : they can appear at the beginning of a plan in any order



Independent Actions

Example: dependent actions

Mr12 (i.e., move(robr,loc1,loc2)) and Lar1 are not independent in A1: a plan starting with Mr12 will be in a state where robr is in loc2, hence Lar1 is not applicable.



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 $\langle a_1, \dots, a_k \rangle$ of the elements of π , the sequence $\langle a_1, \dots, a_k \rangle$ is applicable to s , and the state resulting from the application of π to s is such that

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

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 $\Pi = \langle \pi_1, \dots, \pi_n \rangle$ is a solution to a problem (O, s_0, g) iff :

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

Proposition (Layered Plan Concurrency)

If $\Pi = \langle \pi_1, \dots, \pi_n \rangle$ is a solution plan to a problem (O, s_0, g) , then a sequence of actions corresponding to any permutation of the elements of π_1 , followed by any permutation of π_2, \dots , 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

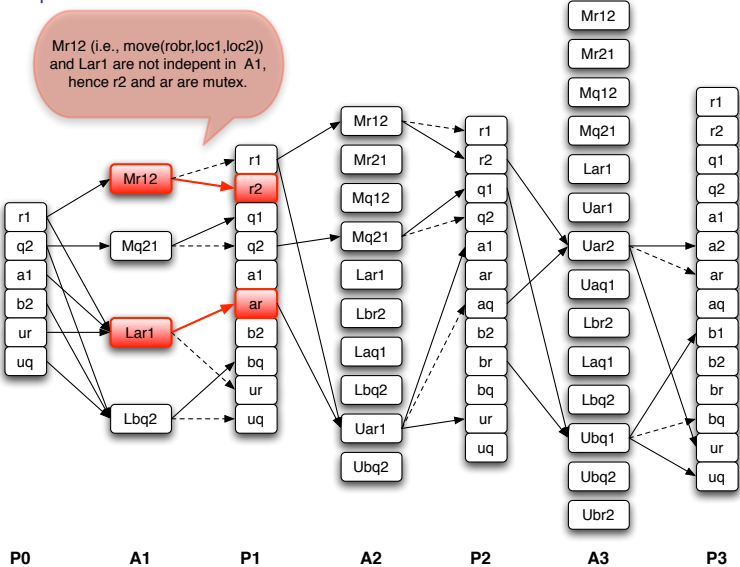
Proposition Mutex

- 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 sense 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

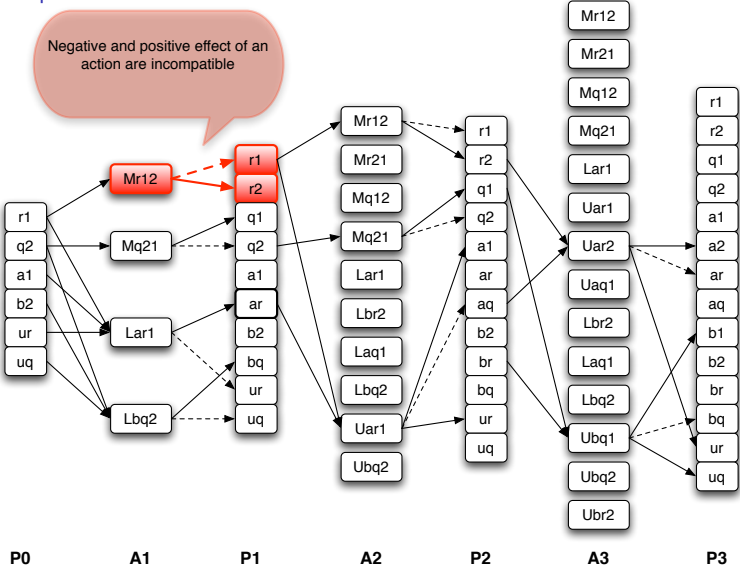
Mr12 (i.e., move(robr,loc1,loc2)) and Lar1 are not independent in A1, hence r2 and ar are mutex.



Mutual Exclusion Relations

Example: Proposition Mutex case 2

Negative and positive effect of an action are incompatible



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 additional incompatible actions in the following level A_{i+1}
- These are actions whose preconditions are incompatible

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 additional constraints of the problem
- For a same problem, a pair 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

Example¹

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	$\{b_r\} \times \{r_1, b_2, u_r, b_q, a_r\}$
...	...

¹A 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 propositions 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-mutex 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 **iterative 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 from the last level of this graph for a solution
- The first extraction, proceeds to level P_i in which all of the goal propositions are included and no pairs of them are mutex
 - ▶ it does not make sense 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** termination 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
 - ★ $\text{precond}(\alpha_p) = \text{effects}^+(\alpha_p)$, and
 - ★ $\text{effects}^-(\alpha_p) = \emptyset$
- A planning graph for a planning problem expanded up to level i is a sequence of layers of nodes and of mutex pairs:

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

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

$A_i \leftarrow \{a \in A \mid \text{precond}(a) \in P_{i-1} \text{ and } \text{precond}(a) \cap \mu P_{i-1} = \emptyset\}$

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

$\mu A_i \leftarrow \{(a, b) \in A_i, a \neq b \mid a, b \text{ are dependent}$
or $\exists (p, q) \in \mu P_{i-1} : p \in \text{precond}(a) \text{ and } q \in \text{precond}(b)\}$

$\mu P_i \leftarrow \{(p, q) \in P_i, p \neq q \mid \forall a, b \in A_i, a \neq b :$
 $p \in \text{effects}^+(a) \text{ and } q \in \text{effects}^+(b) \Rightarrow (a, b) \in \mu A_i\}$

foreach $a \in A_i$ **do**

 link a with a precondition arcs to $\text{precond}(a)$ in P_{i-1}

 link a with a positive arcs to $\text{effects}^+(a)$ in P_i

 link a with a negative arcs to $\text{effects}^-(a)$ in P_i

end

return $\langle P_0, A_1, \mu A_1, \dots, P_{i-1}, \mu P_{i-1}, A_i, \mu A_i, P_i, \mu P_i \rangle$

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 sets 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-point level κ , which is the smallest k such that $|P_{k-1}| = |P_k|$ and $|\mu P_{k-1}| = |\mu P_k|$.

Proof

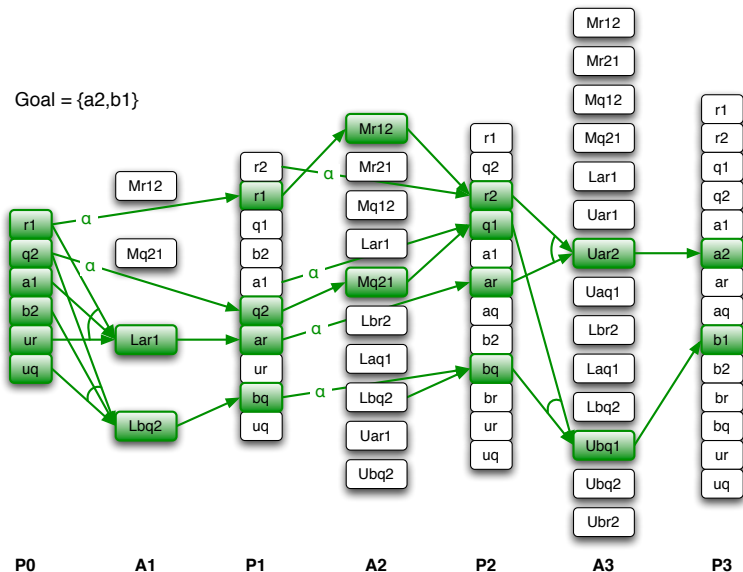
To do ...

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 level $i - 1$ 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 $\langle \pi_1, \dots, \pi_i \rangle$ 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
 $\pi \leftarrow \text{GP-Search}(G, g, \emptyset, i)$ 
if  $\pi \neq \text{failure}$  then return  $\pi$ 
 $\nabla(i) \leftarrow \nabla(i) \cup \{g\}$ 
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 following level $i - 1$

Searching the Planning Graph

GP-Search Procedure

Algorithm (GP-Search(G, g, π, i))

```
if  $g = \emptyset$  then  
   $\Pi \leftarrow \text{Extract}(G, \bigcup\{\text{precond}(a) \mid \forall a \in \pi\}, i - 1)$   
  if  $\Pi = \text{failure}$  then return failure  
  return  $\Pi.\langle\pi\rangle$   
else  
  select any  $p \in g$   
   $\text{resolvers} \leftarrow \{a \in A_i \mid p \in \text{effects}^+(a) \text{ and } \forall b \in \pi : (a, b) \notin \mu A_i\}$   
  if  $\text{resolvers} = \emptyset$  then return failure  
  nondeterministically choose  $a \in \text{resolvers}$   
  return GP-Search( $G, g - \text{effects}^+(a), \pi \cup \{a\}, i$ )  
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 interactive deepening is pursued even after a fixed-point level has been reached until
 - ① success or
 - ② the termination condition is satisfied
 - ★ This termination 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  
   $i \leftarrow i + 1, G \leftarrow \text{Expand}(G, g, i)$   
until [ $g \subseteq P_i$  or  $g \cap \mu P_i = \emptyset$ ] or Fixedpoint( $G$ )  
if  $g \not\subseteq P_i$  or  $g \cap P_i \neq \emptyset$  then return failure  
 $\Pi \leftarrow \text{Extract}(G, g, i)$   
if Fixedpoint( $G$ ) then  $\eta \leftarrow |\nabla(\kappa)|$  else  $\eta \leftarrow 0$   
while  $\Pi = \text{failure}$  do  
   $i \leftarrow i + 1, G \leftarrow \text{Expand}(G, g, i), \Pi \leftarrow \text{Extract}(G, g, i)$   
  if  $\Pi = \text{failure}$  and Fixedpoint( $G$ ) then  
    if  $\eta = |\nabla(\kappa)|$  then return failure  
  end  
   $\eta \leftarrow |\nabla(\kappa)|$   
end  
return  $\Pi$ 
```


Analysis of Graphplan

Soundness, Completeness, and Termination

- 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 Termination

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 it 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 Termination

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 brought 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
- Several techniques and heuristics have been devised to speed-up the search and to improve the memory management of its data structure