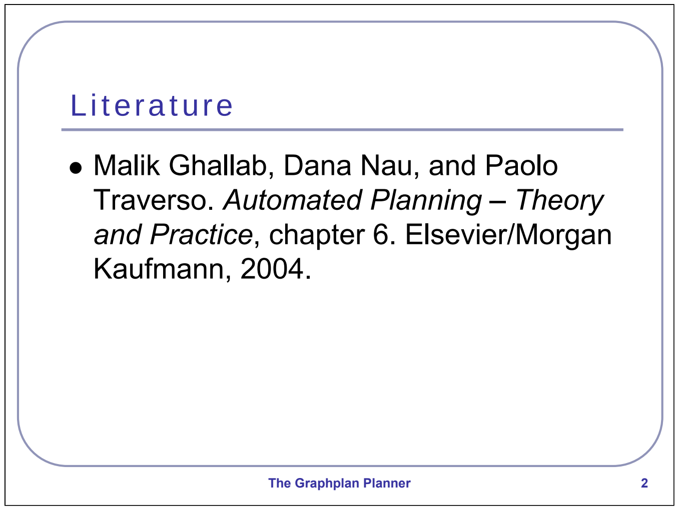
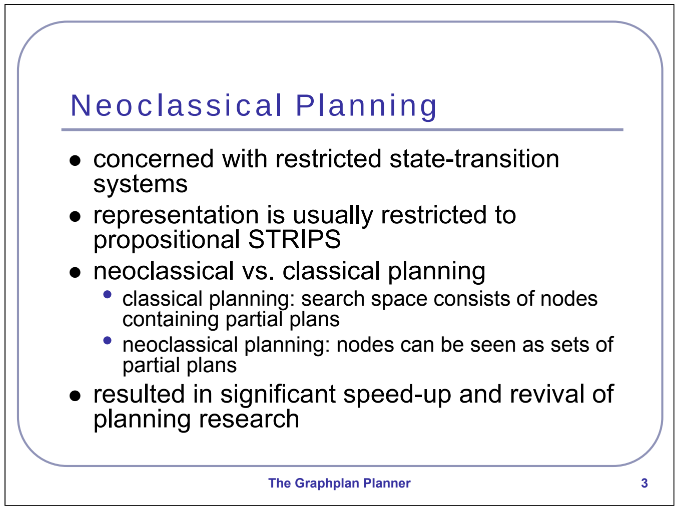
The Graphplan Planner

•Searching the Planning Graph



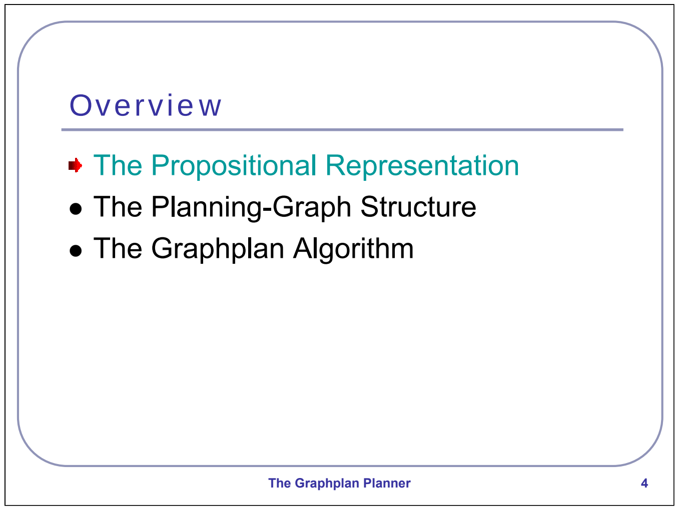
Literature

•Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning – Theory and Practice*, chapter 6. Elsevier/Morgan Kaufmann, 2004.



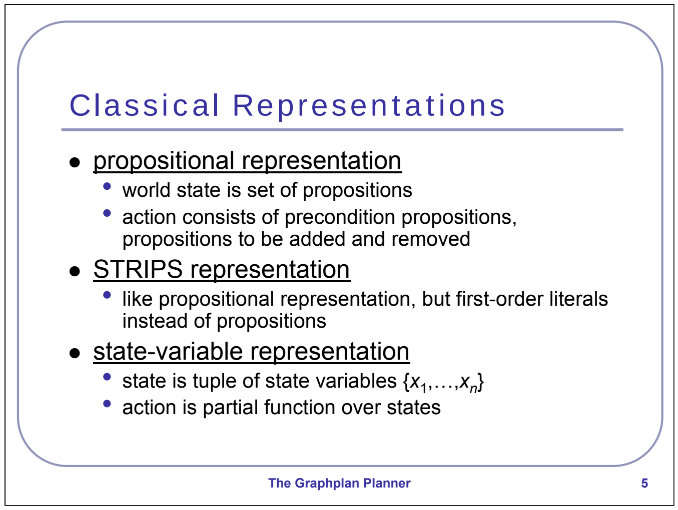
Neoclassical Planning

- **concerned with restricted state-transition systems**
- **representation is usually restricted to propositional STRIPS**
    - no loss in expressive ness due to lack of functions in STRIPS, but loss of potential
- **neoclassical vs. classical planning**
    - **classical planning: search space consists of nodes containing partial plans**
        - every action in a partial plan will appear in the final plan
    - **neoclassical planning: nodes can be seen as sets of partial plans**
        - actions may appear in final plan; disjunctive planning
- **resulted in significant speed-up and revival of planning research**
    - speed-up: blocks world: less than 10 blocks to hundreds



Overview

- **The Propositional Representation**
  - now: the restricted representation used by most neoclassical planning algorithms: propositional STRIPS
- **The Planning-Graph Structure**
- **The Graphplan Algorithm**



Classical Representations

•**<u>propositional representation</u>**

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

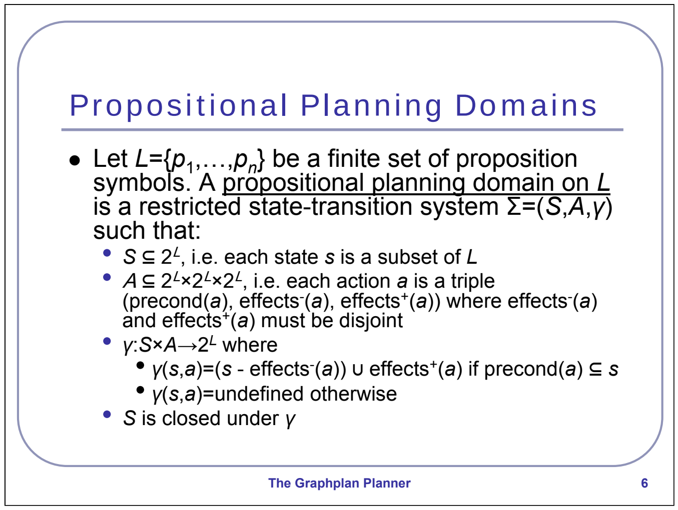
•**<u>STRIPS representation</u>**

- named after STRIPS planner
- **like propositional representation, but first-order literals instead of propositions**
- most popular for restricted state-transitions systems

•**<u>state-variable representation</u>**

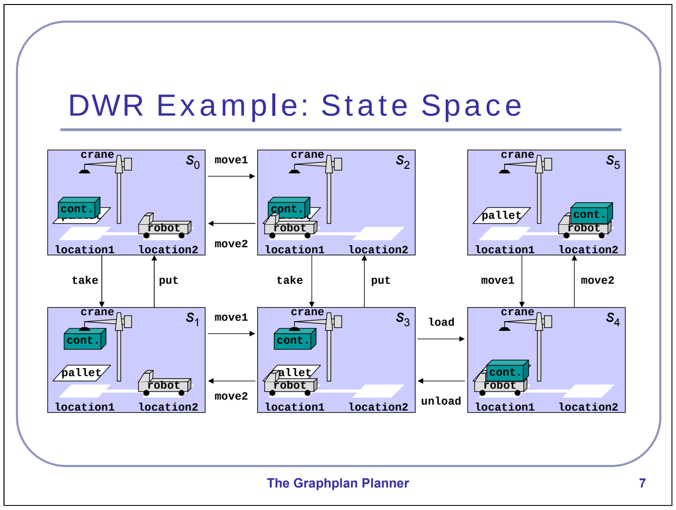
- **state is tuple of state variables $\{x_1,\ldots,x_n\}$**
- **action is partial function over states**
- useful where state is characterized by attributes over finite domains

•equally expressive: planning domain in one representation can also be represented in the others



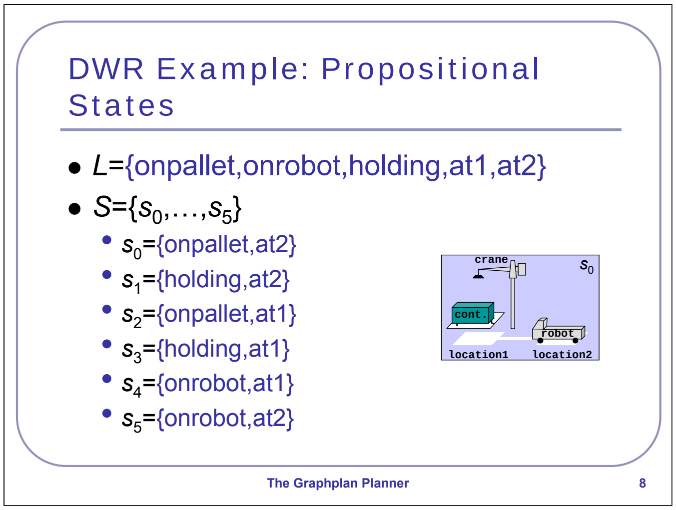
Propositional Planning Domain

- **Let $L=\{p_1,\dots,p_n\}$ be a finite set of proposition symbols. A <u>propositional planning domain</u> 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
    - s is set of propositions that currently hold, i.e. p is true is s iff $p \in s$ (closed world)
  - $A \subseteq 2^L \times 2^L \times 2^L$, i.e. each action a is a triple (precond(a), effects$^-$(a), effects$^+$(a)) where effects$^-$(a) and effects$^+$(a) must be disjoint
    - preconditions, negative effects, and positive effects
    - a is applicable in s iff precond(a) $\subseteq s$
  - **$\gamma: S \times A \to 2^L$ where**
    - $\gamma(s,a)=(s$ - effects$^-(a)) \cup$ effects$^+(a)$ if precond$(a) \subseteq s$
    - **$\gamma(s,a)$=undefined otherwise**
  - **S is closed under γ**
    - if $s \in S$ then for every applicable action a $\gamma(s,a) \in S$



DWR Example: State Space

•from introduction



DWR Example: Propositional States

•*L*={onpallet,onrobot,holding,at1,at2}

- •meaning: container is on the ground, container on the robot, crane is holding the container, robot is at location1, robot is at location2

•*S*={s_0,…,s_5}

- •as shown in graph
- **•s_0={onpallet,at1}**
- **•s_1={holding,at1}**
- **•s_2={onpallet,at1}**
- **•s_3={holding,at1}**
- **•s_4={onrobot,at1}**
- **•s_5={onrobot,at2}**

DWR Example: Propositional Actions

a	precond(*a*)	effects$^-$(*a*)	effects$^+$(*a*)
take	{onpallet}	{onpallet}	{holding}
put	{holding}	{holding}	{onpallet}
load	{holding,at1}	{holding}	{onrobot}
unload	{onrobot,at1}	{onrobot}	{holding}
move1	{at2}	{at2}	{at1}
move2	{at1}	{at1}	{at2}

DWR Example: Propositional Actions

- ***a*** : precond(*a*), effects$^-$(*a*), effects$^+$(*a*)
  - *a* is action name
- **take : {onpallet}, {onpallet}, {holding}**
- **put : {holding}, {holding}, {onpallet}**
- **load : {holding,at1}, {holding}, {onrobot}**
- **unload : {onrobot,at1}, {onrobot}, {holding}**
- **move1 : {at2}, {at2}, {at1}**
- **move2 : {at1}, {at1}, {at2}**

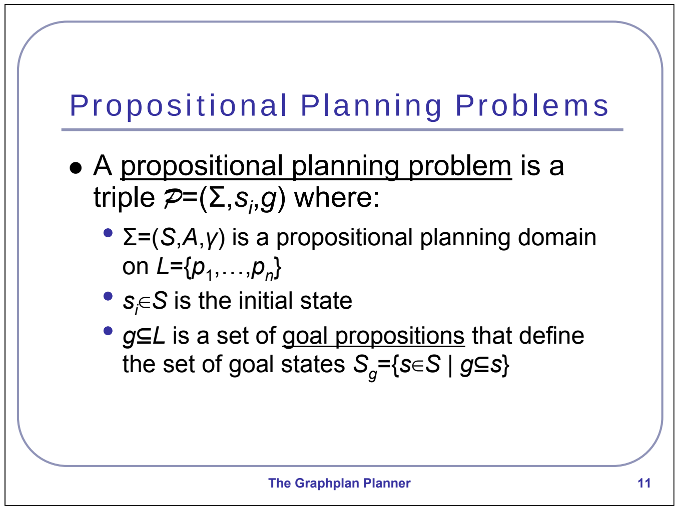
DWR Example: Propositional State Transitions

	s_0	s_1	s_2	s_3	s_4	s_5
take	s_1		s_3			
put		s_0		s_2		
load				s_4		
unload					s_3	
move1			s_0	s_1		s_4
move2	s_2	s_3			s_5	

DWR Example: Propositional State Transitions

•columns: action *a*; rows: state *s*; table cell entry: $\gamma(s,a)$ or empty if action not applicable

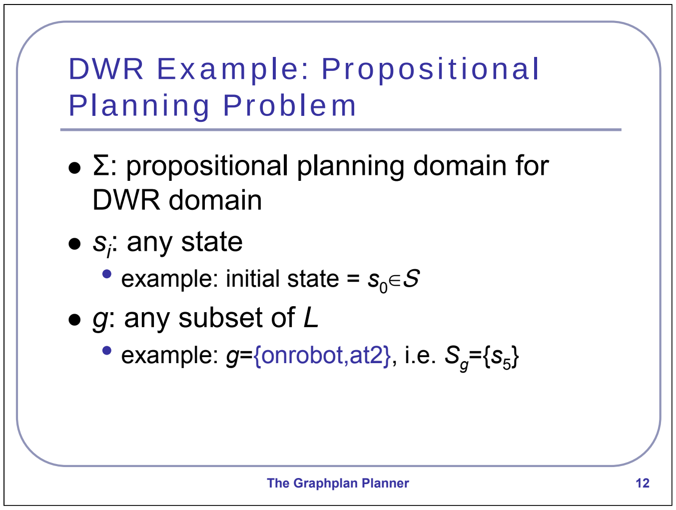
•example: $\gamma(s_0,\text{take})=s_1$



Propositional Planning Problems

•A <u>propositional planning problem</u> is a triple $\mathcal{P}=(\Sigma,s_i,g)$ 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 <u>goal propositions</u> that define the set of goal states $S_g=\{s \in S \mid g \subseteq s\}$
  - •gaol states are implicit in the problem



DWR Example: Propositional Planning Problem

•Σ: propositional planning domain for DWR domain

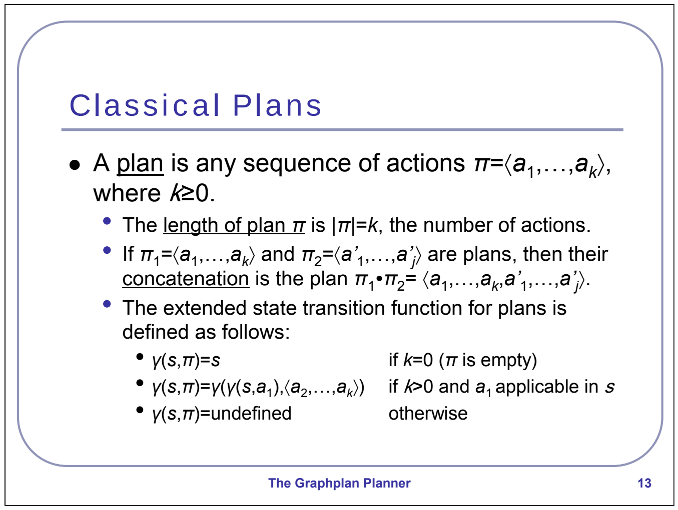
- •see previous slides

•s_i: any state

- **•example: initial state = $s_0 \in S$**
- •note: s_0 is not necessarily initial state

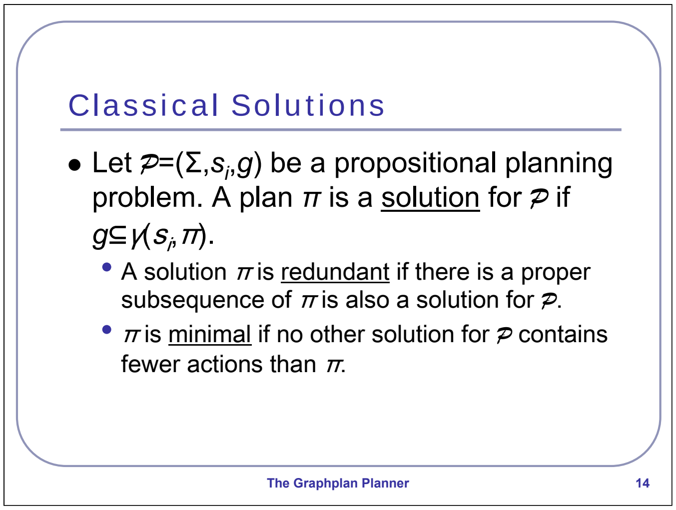
•g: any subset of L

- **•example: g={onrobot,at2}, i.e. $S_g=\{s_5\}$**



Classical Plans

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



Classical Solutions

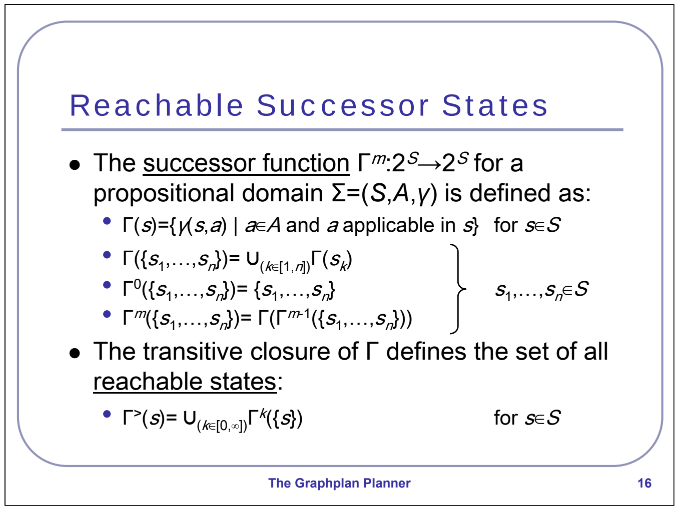
- note: exactly as for STRIPS case
- **Let $\mathcal{P}=(\Sigma,s_i,g)$ be a propositional planning problem. A plan π is a <u>solution</u> for $\mathcal{P}$ if $g \subseteq \gamma(s_i,\pi)$.**
    - A solution π is <u>redundant</u> if there is a proper subsequence of π is also a solution for $\mathcal{P}$.
    - π is <u>minimal</u> if no other solution for $\mathcal{P}$ contains fewer actions than π.

DWR Example: Plans and Solutions

plan π	$\|\pi\|$	$\gamma(s_i,\pi)$	sol.	red.	min.
⟨⟩	0	s_0	no	-	-
⟨move2,move2⟩	2	undef.	no	-	-
⟨take,move1⟩	2	s_3	no	-	-
⟨take,move1,put,move2, take,move1,load,move2⟩	8	s_5	yes	yes	no
⟨take,move1,load,move2⟩	4	s_5	yes	no	yes
⟨move1,take,load,move2⟩	4	s_5	yes	no	yes

DWR Example: Plans and Solutions

- as before: s_i=s_0; g={onrobot,at2}, i.e. S_g={s_5}



Reachable Successor States

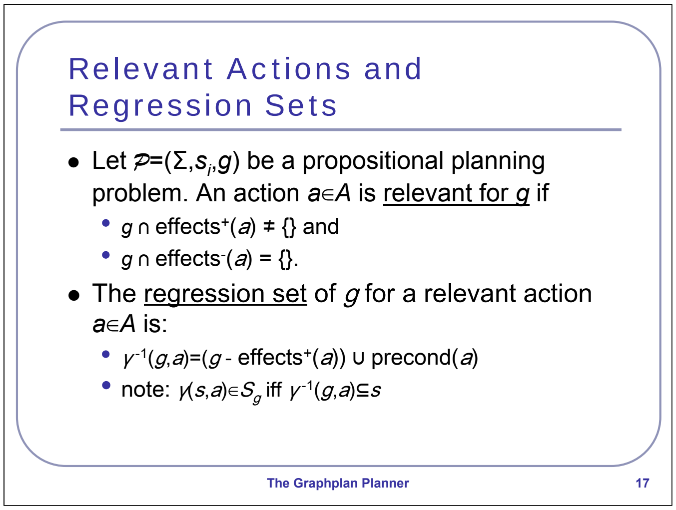
•note: exactly as for STRIPS case

•**The <u>successor function</u>** $\Gamma^m:2^S\rightarrow 2^S$ **for a propositional domain $\Sigma=(S,A,\gamma)$ is defined as:**

- •$\Gamma(s)=\{\gamma(s,a) \mid a\in A$ and a applicable in $s\}$ for $s\in S$
- •$\Gamma(\{s_1,\ldots,s_n\})= \cup_{(k\in[1,n])}\Gamma(s_k)$
- •$\Gamma^0(\{s_1,\ldots,s_n\})= \{s_1,\ldots,s_n\}$
- •$\Gamma^m(\{s_1,\ldots,s_n\})= \Gamma(\Gamma^{m-1}(\{s_1,\ldots,s_n\}))$

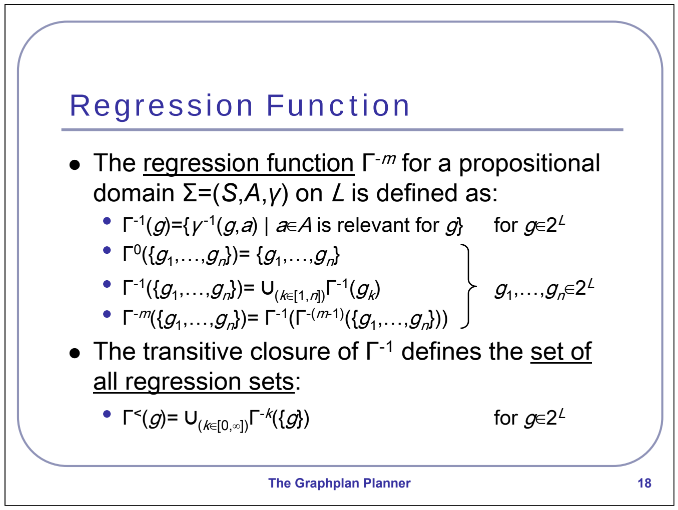
•The transitive closure of Γ defines the set of all <u>reachable states</u>:

- •$\Gamma^>(s)= \cup_{(k\in[0,\infty])}\Gamma^k(\{s\})$ for $s\in S$



Relevant Actions and Regression Sets

- **Let $\mathcal{P}=(\Sigma,s_i,g)$ be a propositional planning problem. An action $a \in A$ is <u>relevant for g</u> if**
    - $g \cap \text{effects}^+(a) \neq \{\}$ and
    - $g \cap \text{effects}^-(a) = \{\}$.
    - intuition: a is relevant for g if it can contribute toward producing a state in S_g
- The <u>regression set</u> of g for a relevant action $a \in A$ **is:**
    - $\gamma^{-1}(g,a)=(g - \text{effects}^+(a)) \cup \text{precond}(a)$
    - $\mathcal{P}=(\Sigma,s_i,g)$ has a solution if $\exists a \in A : \mathcal{P}=(\Sigma,s_i,\gamma^{-1}(g,a))$
    - note: $\gamma(s,a) \in S_g$ iff $\gamma^{-1}(g,a) \subseteq s$
    - $\gamma^{-1}(g,a)$: minimal set of propositions that must hold in a state s from which action a leads to a goal state



Regression Function

•note: exactly as for STRIPS case

•**The <u>regression function</u>** Γ^{-m} for a propositional domain $\Sigma=(S,A,\gamma)$ on L is defined as:

- $\Gamma^{-1}(g)=\{\gamma^{-1}(g,a) \mid a\in A$ is relevant for $g\}$ for $g\in 2^L$
- $\Gamma^{0}(\{g_1,\ldots,g_n\})= \{g_1,\ldots,g_n\}$
- $\Gamma^{-1}(\{g_1,\ldots,g_n\})= \cup_{(k\in[1,n])}\Gamma^{-1}(g_k)$
- $\Gamma^{-m}(\{g_1,\ldots,g_n\})= \Gamma^{-1}(\Gamma^{-(m-1)}(\{g_1,\ldots,g_n\}))$

•The transitive closure of Γ^{-1} defines the <u>set of all regression sets</u>:

- $\Gamma^{<}(g)= \cup_{(k\in[0,\infty])}\Gamma^{-k}(\{g\})$ for $g\in 2^L$

Statement of a Propositional Planning Problem

- A <u>statement of a propositional planning problem</u> 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 $\mathcal{P}=(\Sigma,s_i,g)$
  - g is a set of goal propositions in the same propositional planning problem $\mathcal{P}$

The Graphplan Planner 19

Statement of a Propositional Planning Problem

•A <u>statement of a propositional planning problem</u> is a triple $P=(A,s_i,g)$ where:

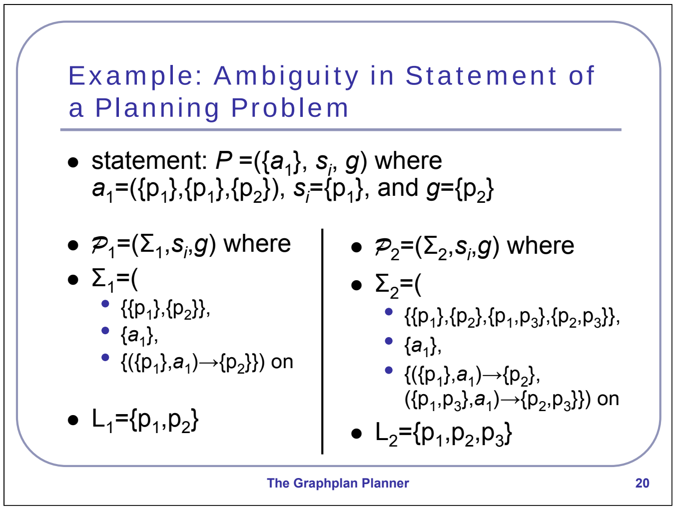
•A is a set of actions in an appropriate propositional planning domain $\Sigma=(S,A,\gamma)$ on L

•s_i is the initial state in an appropriate propositional planning problem $\mathcal{P}=(\Sigma,s_i,g)$

•g is a set of goal propositions **in the same propositional planning problem $\mathcal{P}$**

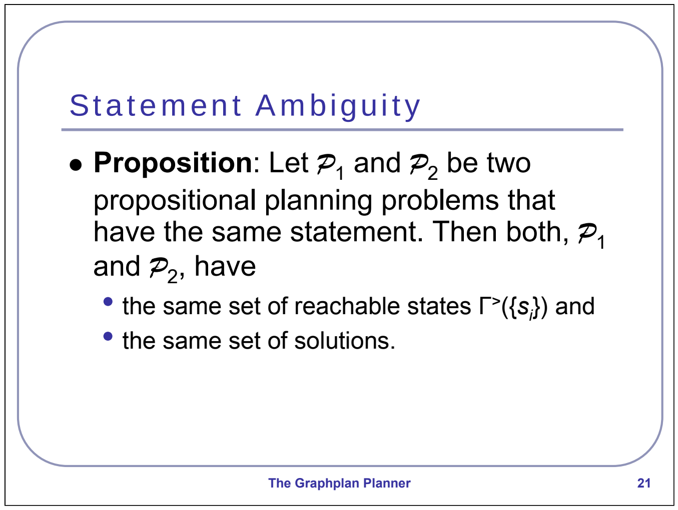
•*advantage: statement does not require explicit enumeration of S and γ*

•problem: L, S *and* γ are ambiguous



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}**
- P is statement of planning problem:
  - **$\mathcal{P}_1=(\Sigma_1,s_i,g)$ where**
  - Σ_1=(**{{p_1},{p_2}}, {a_1}, {({p_1},a_1)→{p_2}}**) on
  - **L_1={p_1,p_2}**
- alternative:
  - **$\mathcal{P}_2=(\Sigma_2,s_i,g)$ where**
  - Σ_2=(**{{p_1},{p_2},{p_1,p_3},{p_2,p_3}}, {a_1}, {({p_1},a_1)→{p_2}, ({p_1,p_3},a_1)→{p_2,p_3}}**) on
  - **L_2={p_1,p_2,p_3}**
  - p_3 plays no role in $\mathcal{P}_2$
- regression sets $\Gamma^<$({g}) and reachable states $\Gamma^>$({s_i}) are identical in $\mathcal{P}_1$ and $\mathcal{P}_2$

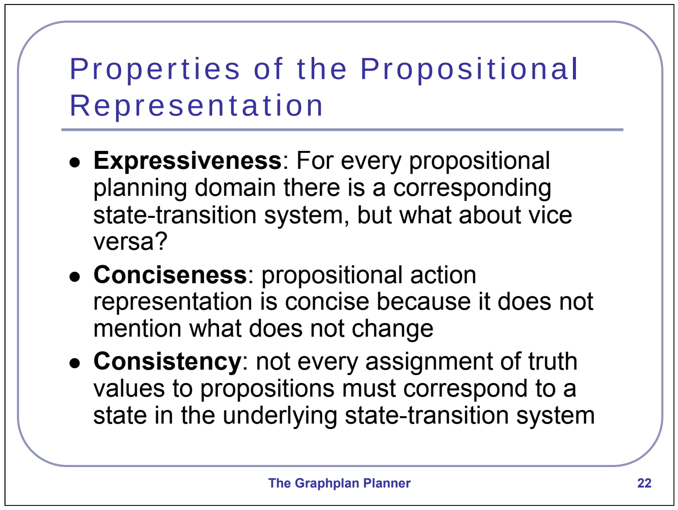


Statement Ambiguity

•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 $\Gamma^>(\{s_i\})$ and**
- **•the same set of solutions.**

•statements are unambiguous enough to be acceptable specifications of planning problems



Properties of the Propositional Representation

•Expressiveness: For every propositional planning domain there is a corresponding state-transition system, but what about vice versa?

- •depends on definition of “corresponding”

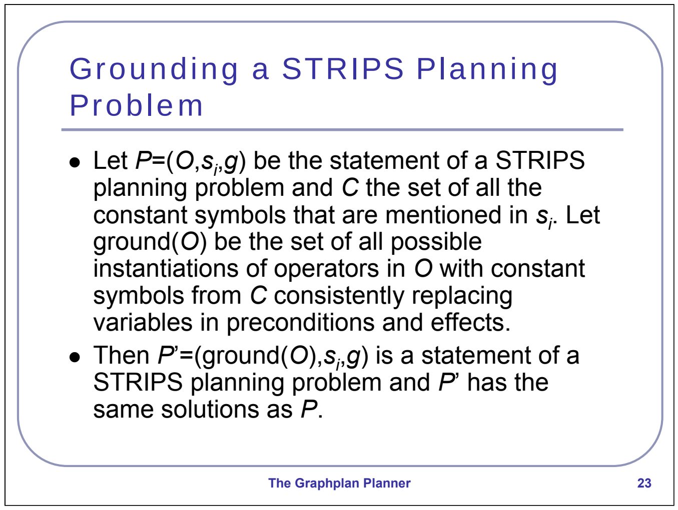
•Conciseness: propositional action representation is concise because it does not mention what does not change

- •truth values of propositions not mentioned in an action do not change through the application of the action, they persist

•Consistency: not every assignment of truth values to propositions must correspond to a state in the underlying state-transition system

- •example from DWR domain: state {onrobot,holding,at1,at2} is inconsistent
- •if domain definition and initial state are correct, inconsistent states should not be reachable

•note: state-space and plan-space search still applicable



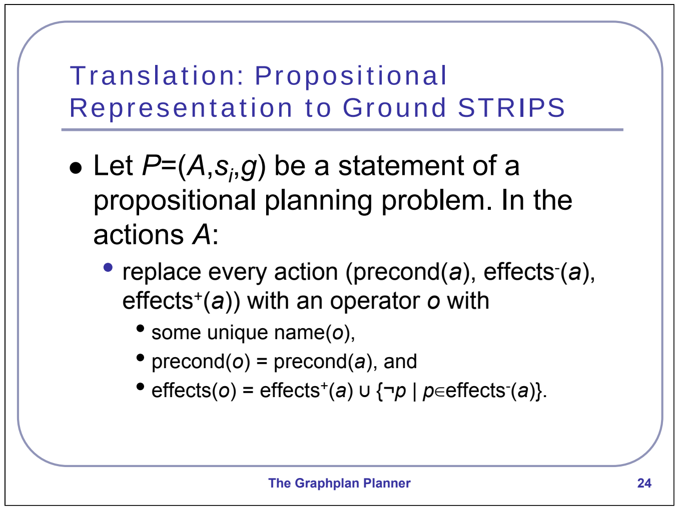
Grounding a STRIPS Planning Problem

•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 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.

- •the number of operators will increase exponentially here

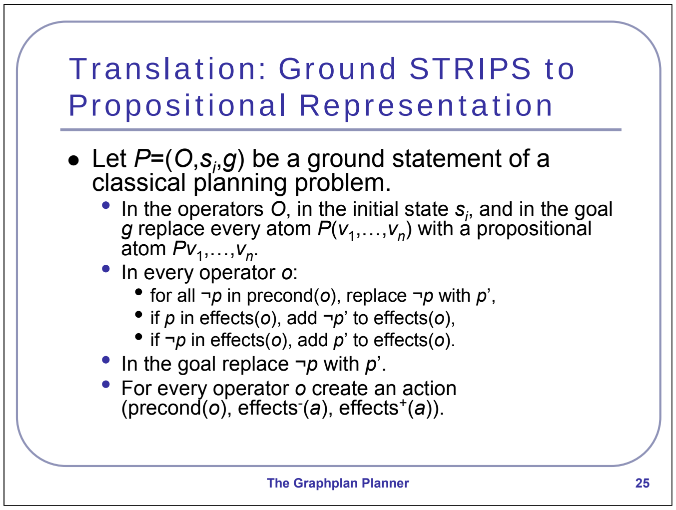
•Then P'=(ground(O),s_i,g) is a statement of a STRIPS planning problem and P' has the same solutions as P.

- •the problems are equivalent (except for exponential increase in size)



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 (precond(a), effects$^-$(a), effects$^+$(a)) with an operator o with**
        - **some unique name(o),**
        - **precond(o) = precond(a), and**
        - **effects(o) = effects$^+$(a) ∪ {¬p | p∈effects$^-$(a)}.**
            - adds negation sign to negative effects
- result is a statement of a ground STRIPS planning problem

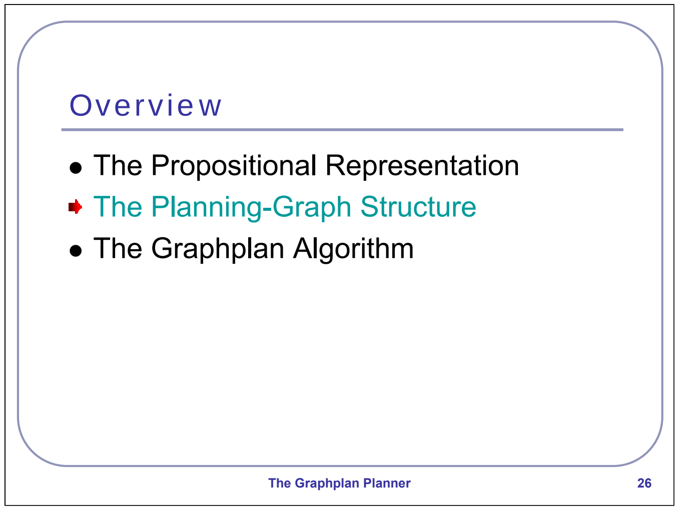


Translation: Ground STRIPS to Propositional Representation

•Let P=(O,s_i,g) be a ground statement of a classical planning problem.

- •problem: operators may contain negated preconditions
- **•In the operators O, in the initial state s_i, and in the goal g replace every atom $P(v_1,\ldots,v_n)$ with a propositional atom $Pv_1,\ldots,v_n$.**
- •idea: introduce new proposition symbols that represent the negations of existing propositions
- **•In every operator o:**
  - **•for all ¬p in precond(o), replace ¬p with p',**
  - **•if p in effects(o), add ¬p' to effects(o),**
  - **•if ¬p in effects(o), add p' to effects(o).**
- **•In the goal replace ¬p with p'.**
- **•For every operator o create an action (precond(o), effects$^-$(a), effects$^+$(a)).**

•result is a statement of a propositional planning problem



Overview

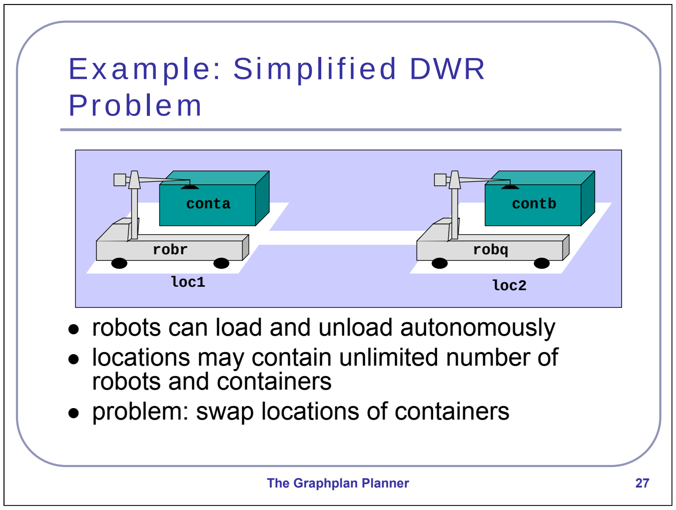
➧**The Propositional Representation**

- ➧just done: the restricted representation used by most neoclassical planning algorithms: propositional STRIPS

•**The Planning-Graph Structure**

- •now: defining a new graph that is more efficient to generate and a necessary criterion for solution containment

•**The Graphplan Algorithm**



Example: Simplified DWR Problem

•[figure]

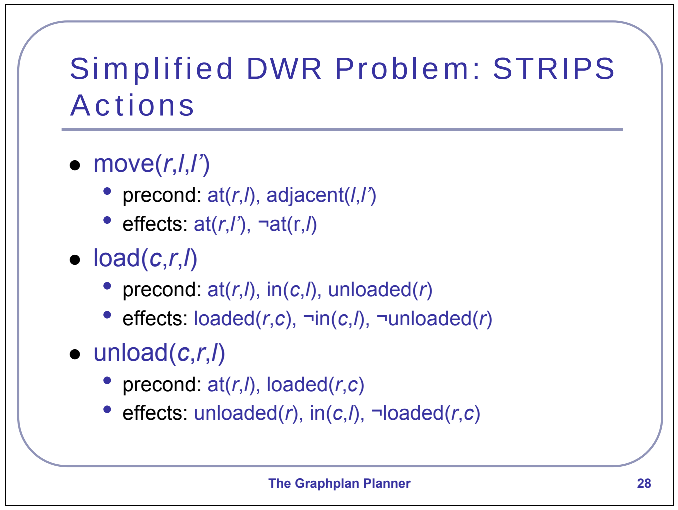
•initial state:

- •2 locations: loc1 and loc2, connected by path
- •2 robots: robr and robq, both unloaded initially at loc1 and loc2 respectively
- •2 containers: conta and contb, initially at loc1 and loc2 respectively

•robots can load and unload autonomously

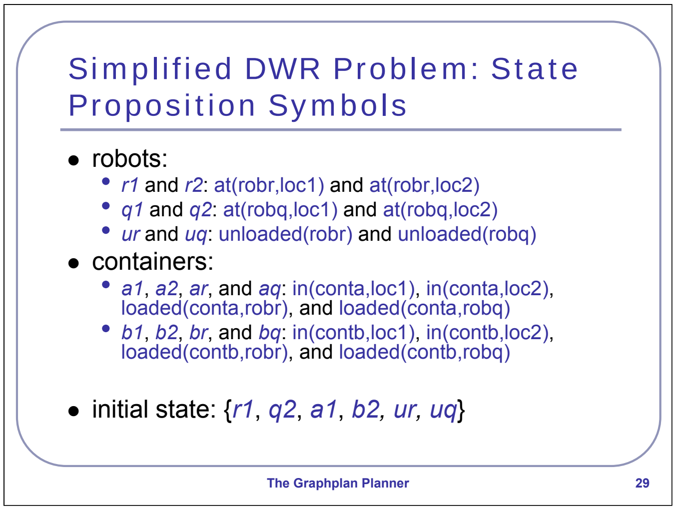
•locations may contain unlimited number of robots and containers

•problem: swap locations of containers



Simplified DWR Problem: STRIPS Actions

- **move(*r*,*l*,*l’*)**
  - move robot *r* from location *l* to adjacent location *l’* (4 possible actions; with rigid adjacent relation evaluated)
  - **precond: at(*r*,*l*), adjacent(*l*,*l’*)**
  - **effects: at(*r*,*l’*), ¬at(r,*l*)**
- **load(*c*,*r*,*l*)**
  - load container *c* onto robot *r* at location *l* (8 possible actions)
  - **precond: at(*r*,*l*), in(*c*,*l*), unloaded(*r*)**
  - **effects: loaded(*r*,*c*), ¬in(*c*,*l*), ¬unloaded(*r*)**
- **unload(*c*,*r*,*l*)**
  - unload container *c* from robot *r* at location *l* (8 possible actions)
  - **precond: at(*r*,*l*), loaded(*r*,*c*)**
  - **effects: unloaded(*r*), in(*c*,*l*), ¬loaded(*r*,*c*)**



Simplified DWR Problem: State Proposition Symbols

•idea: represent each atom that may occur in a state by a single (short) proposition symbol

•**robots:**

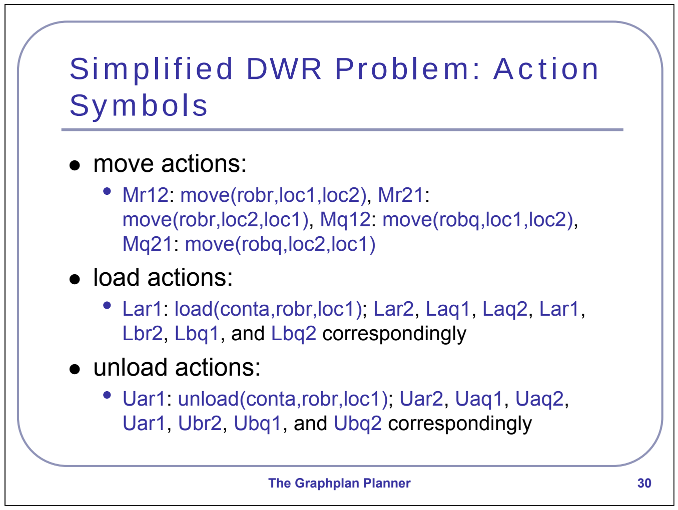
- •***r1*** **and** ***r2*****: at(robr,loc1) and at(robr,loc2)**
- •***q1*** **and** ***q2*****: at(robq,loc1) and at(robq,loc2)**
- •***ur*** **and** ***uq*****: unloaded(robr) and unloaded(robq)**

•**containers:**

- •***a1*****,** ***a2*****,** ***ar*****, and** ***aq*****: in(conta,loc1), in(conta,loc2), loaded(conta,robr), and loaded(conta,robq)**
- •***b1*****,** ***b2*****,** ***br*****, and** ***bq*****: in(contb,loc1), in(contb,loc2), loaded(contb,robr), and loaded(contb,robq)**

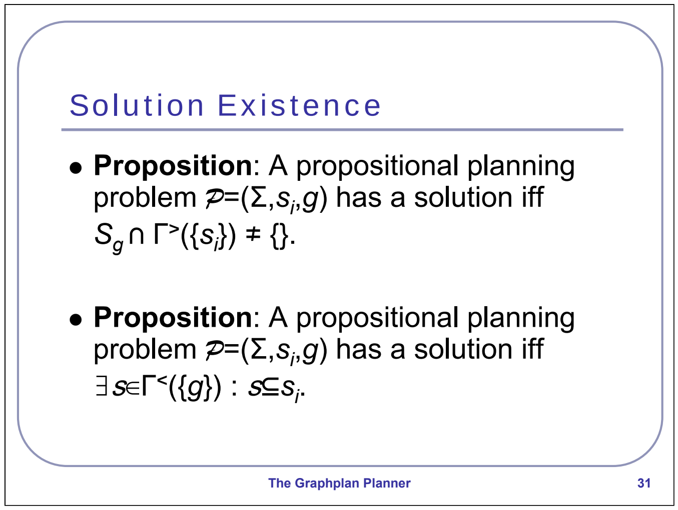
•14 state propositions

•**initial state: {*r1*, *q2*, *a1*, *b2, ur, uq*}**



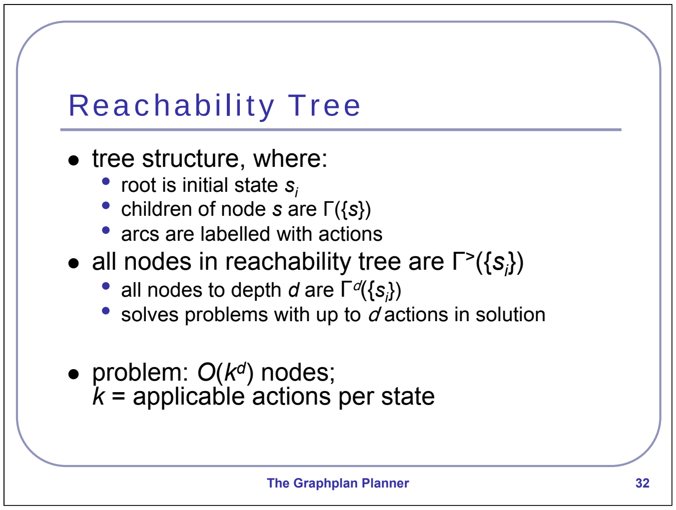
Simplified DWR Problem: Action Symbols

- **move actions:**
  - **Mr12: move(robr,loc1,loc2), Mr21: move(robr,loc2,loc1), Mq12: move(robq,loc1,loc2), Mq21: move(robq,loc2,loc1)**
- **load actions:**
  - **Lar1: load(conta,robr,loc1); Lar2, Laq1, Laq2, Lar1, Lbr2, Lbq1, and Lbq2 correspondingly**
- **unload actions:**
  - **Uar1: unload(conta,robr,loc1); Uar2, Uaq1, Uaq2, Uar1, Ubr2, Ubq1, and Ubq2 correspondingly**
- 14 state symbols: lower case, italic
- 20 action symbols: uppercase, not italic



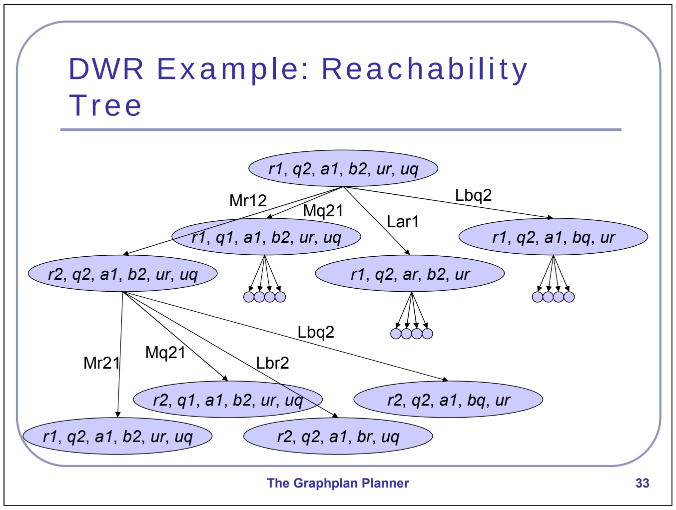
Solution Existence

- **Proposition: A propositional planning problem $\mathcal{P}=(\Sigma,s_i,g)$ has a solution iff $S_g \cap \Gamma^{>}(\{s_i\}) \neq \{\}$.**
    - … iff there is a goal state that is also a reachable state
- **Proposition: A propositional planning problem $\mathcal{P}=(\Sigma,s_i,g)$ has a solution iff $\exists s \in \Gamma^{<}(\{g\}) : s \subseteq s_i$.**
    - … iff there is a minimal set of propositions amongst all regression sets that is a subset of the initial state



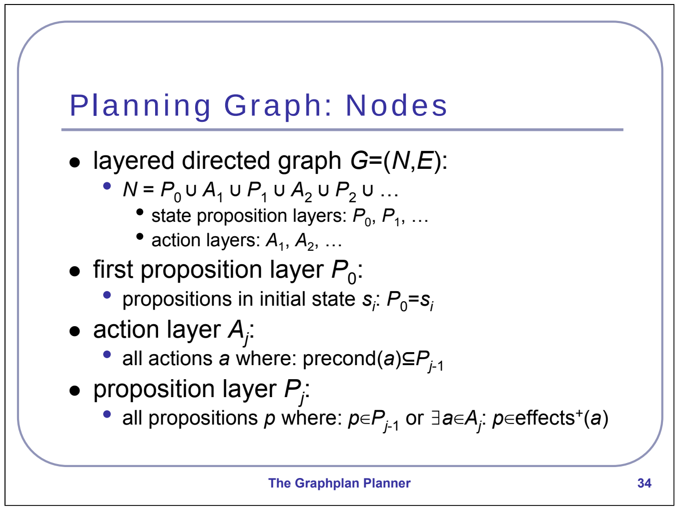
Reachability Tree

- **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



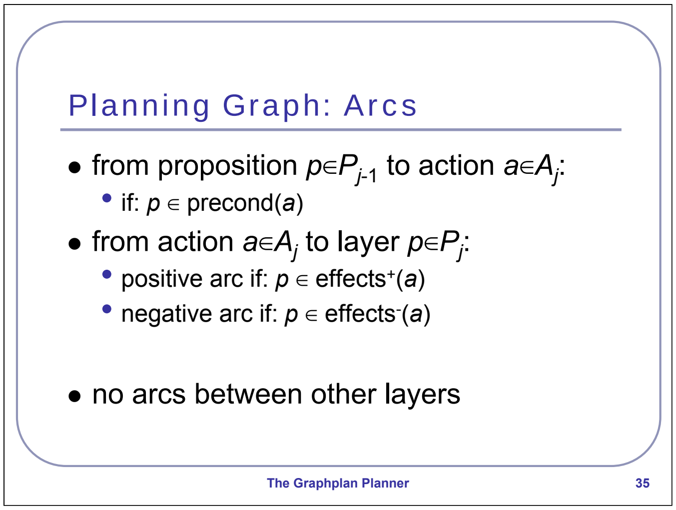
DWR Example: Reachability Tree

- **[figure]**
- corresponds directly to forward-search search tree
- actually: should be graph (corresponding to state space)



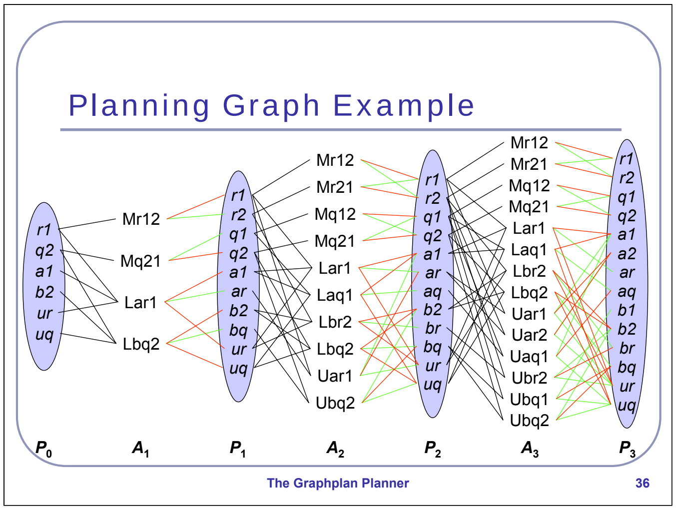
Planning Graph: Nodes

- **layered directed graph $G=(N,E)$:**
  - layered = each node belongs to exactly one layer
  - **$N = P_0 \cup A_1 \cup P_1 \cup A_2 \cup P_2 \cup \ldots$**
    - proposition and action layers alternate
    - **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: precond(a)$\subseteq P_{j-1}$**
- **proposition layer P_j:**
  - **all propositions p where: $p \in P_{j-1}$ or $\exists a \in A_j$: $p \in$ effects$^+$(a)**
  - propositions at layer P_j are all propositions in the union of all nodes in the reachability tree at depth j
    - note: negative effects are not deleted from next layer
- note: $P_{j-1} \subseteq P_j$; propositions in the graph monotonically increase from one proposition layer to the next



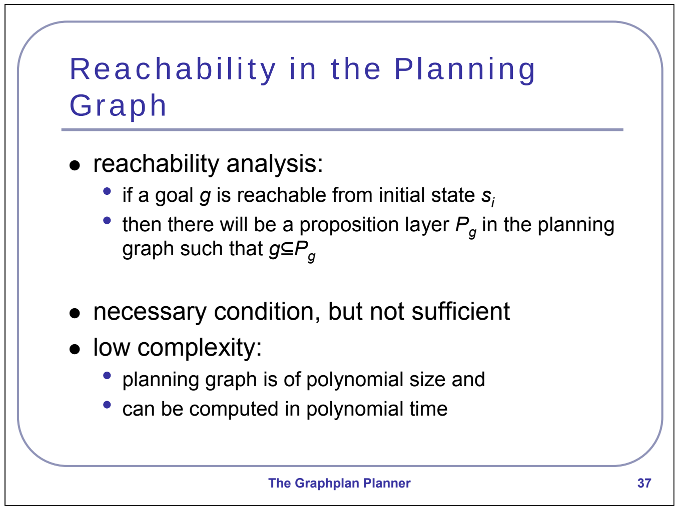
Planning Graph: Arcs

- directed and layered = arcs only from one layer to the next
- **from proposition $p \in P_{j-1}$ to action $a \in A_j$:**
  - **if: $p \in$ precond(a)**
- **from action $a \in A_j$ to layer $p \in P_j$:**
  - **positive arc if: $p \in$ effects$^+$(a)**
  - **negative arc if: $p \in$ effects$^-$(a)**
- **no arcs between other layers**
- note: $A_{j-1} \subseteq A_j$; actions in the graph monotonically increase from one action layer to the next



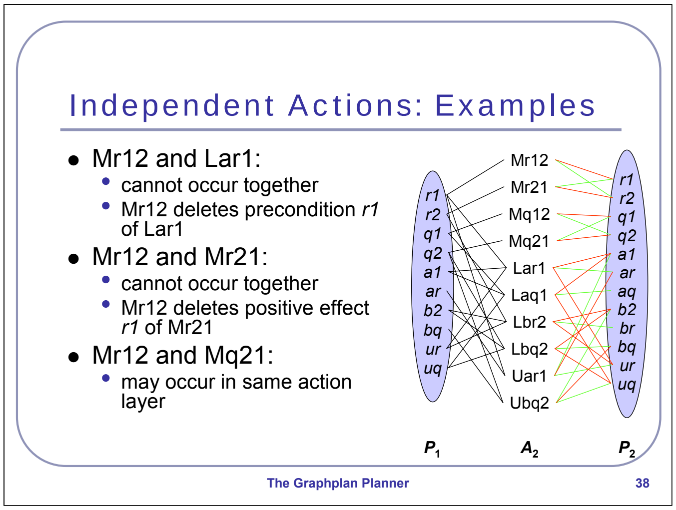
Planning Graph Example

- **[figure]**
    - start with initial proposition layer
    - next action layer: applicable action; links from preconditions (black)
    - next proposition layer: previous proposition plus positive effects; links to positive effects (green); links to negative effects (red)
    - next action layer (A_2); precondition links; next proposition layer (P_2); effect links
    - next action layer (A_3); precondition links; next proposition layer (P_3); effect links
- action layers contain “inclusive disjunctions” of actions



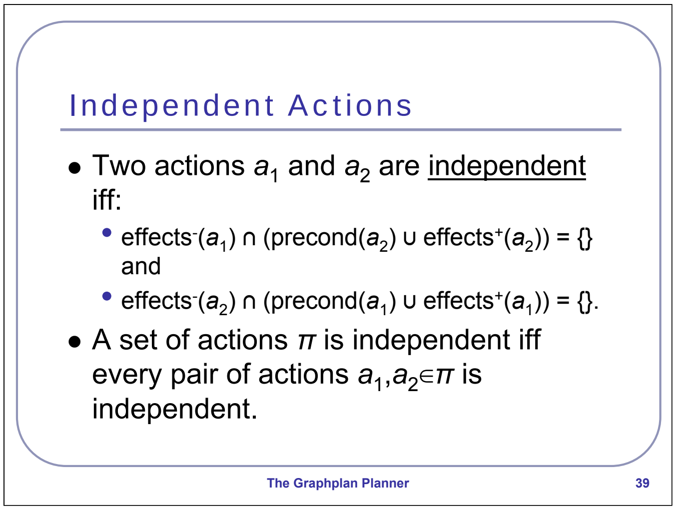
Reachability in the Planning Graph

- **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 $g \subseteq P_g$**
    - or: if no proposition layer contains *g* then *g* is not reachable
- **necessary condition, but not sufficient**
    - necessary vs. sufficient:
        - reachability tree:
            - nodes contain propositions that must necessarily hold
            - propositions in one node are consistent
        - planning graph:
            - proposition layers contains propositions that may possibly hold
            - propositions in one layer usually inconsistent (e.g. robots/containers in two places at once)
            - similarly, incompatible actions in one layer may interfere with each other
- **low complexity:**
    - **planning graph is of polynomial size and**
    - **can be computed in polynomial time**
- need more conditions (for sufficient criterion)



Independent Actions: Examples

- **Mr12 and Lar1:**
    - **cannot occur together**
    - **Mr12 deletes precondition *r1* of Lar1**
- **Mr12 and Mr21:**
    - **cannot occur together**
    - **Mr12 deletes positive effect *r1* of Mr21**
- **Mr12 and Mq21:**
    - **may occur in same action layer**



Independent Actions

•idea: independent actions can be executed in any order (in same layer)

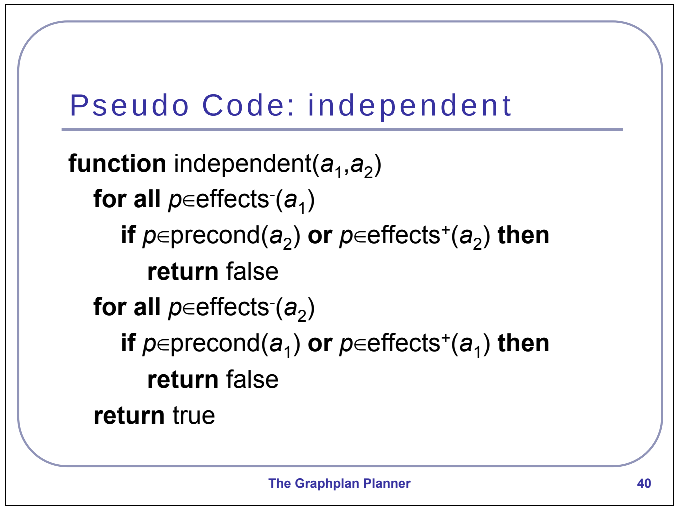
•**Two actions a_1 and a_2 are <u>independent</u> iff:**

- **effects$^-$(a_1) ∩ (precond(a_2) ∪ effects$^+$(a_2)) = {} and**
- **effects$^-$(a_2) ∩ (precond(a_1) ∪ effects$^+$(a_1)) = {}.**
- two actions are dependent iff:
  - one deletes a precondition of the other or
  - one deletes a positive effect of the other

•**A set of actions π is independent iff every pair of actions $a_1,a_2 \in \pi$ is independent.**

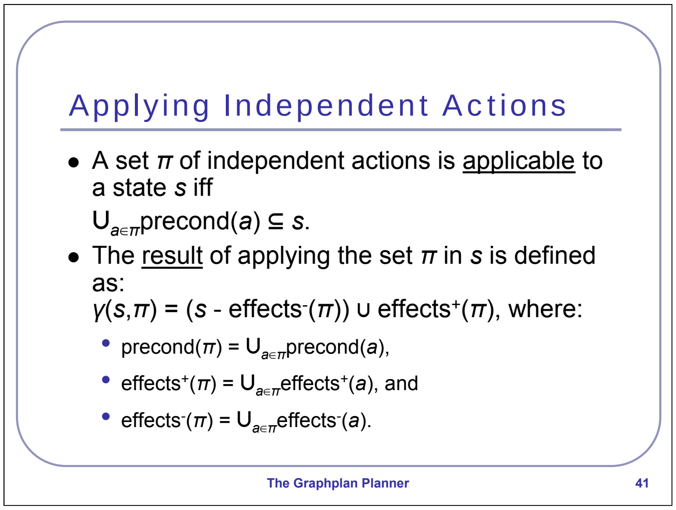
•note: independence does not depend on planning problem; can be pre-computed

•note: independence relation is symmetrical (follows from definition)



Pseudo Code: independent

- **function independent(a_1,a_2)**
    - returns true iff the two given actions are independent
- **for all $p \in$ effects$^-$(a_1)**
- **if $p \in$ precond(a_2) or $p \in$ effects$^+$(a_2) then**
- **return false**
- **for all $p \in$ effects$^-$(a_2)**
- **if $p \in$ precond(a_1) or $p \in$ effects$^+$(a_1) then**
- **return false**
- **return true**
- complexity:
    - let b be max. number of preconditions, positive, and negative effects of any action
    - element test in hash-set takes constant time
    - complexity: $O(b)$



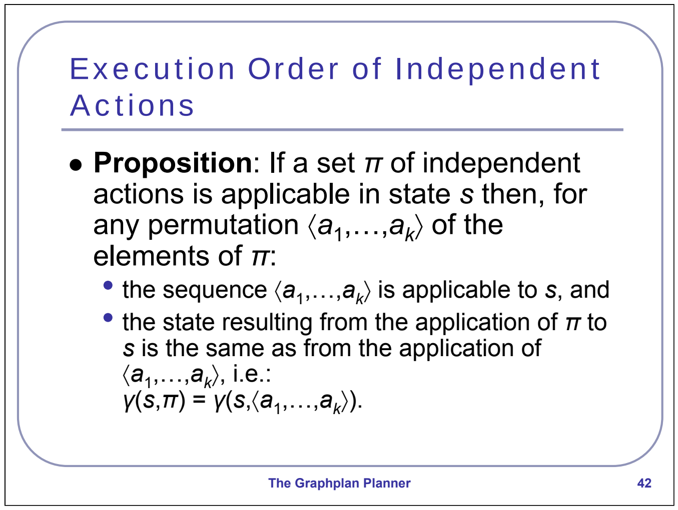
Applying Independent Actions

•A set π of independent actions is <u>applicable</u> to a state s iff $\bigcup_{a\in\pi}\text{precond}(a) \subseteq s$.

•note: applying a set of independent actions can be done in any order

•The <u>result</u> of applying the set π in s is defined as:
$\gamma(s,\pi) = (s - \text{effects}^-(\pi)) \cup \text{effects}^+(\pi)$, where:

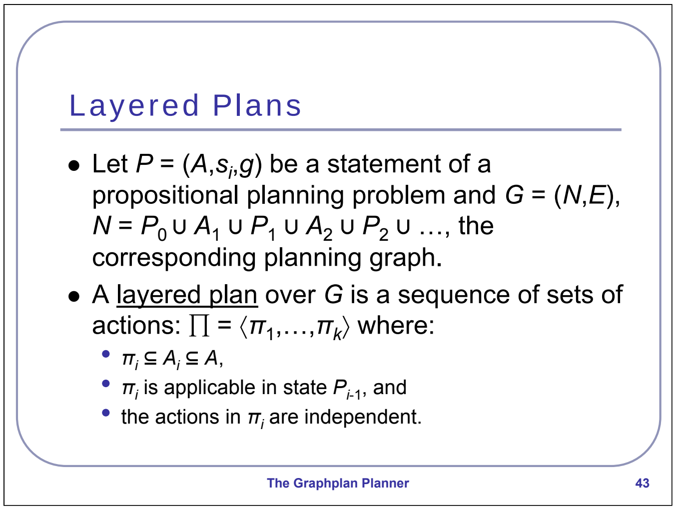
- **•$\text{precond}(\pi) = \bigcup_{a\in\pi}\text{precond}(a)$,**
- **•$\text{effects}^+(\pi) = \bigcup_{a\in\pi}\text{effects}^+(a)$, and**
- **•$\text{effects}^-(\pi) = \bigcup_{a\in\pi}\text{effects}^-(a)$.**



Execution Order of Independent Actions

•Proposition: If a set π of independent actions is applicable in state 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 the same as from the application of $\langle a_1,\ldots,a_k\rangle$, i.e.:**
  $\gamma(s,\pi) = \gamma(s,\langle a_1,\ldots,a_k\rangle)$.

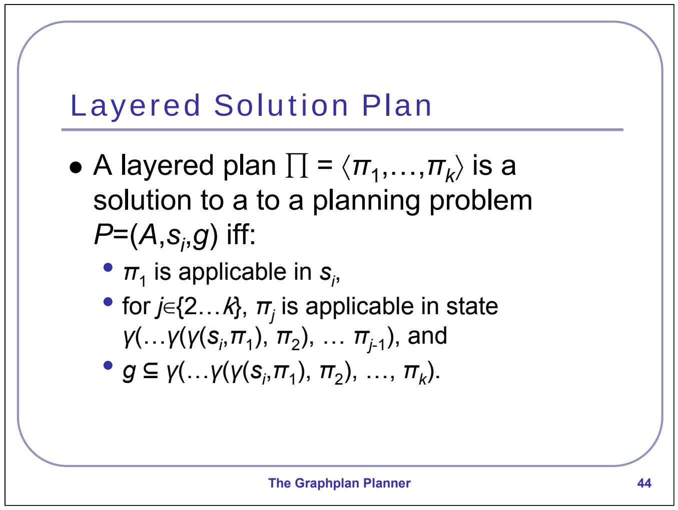


Layered Plans

•Let $P = (A,s_i,g)$ 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 <u>layered plan</u> over G is a sequence of sets of actions: $\prod = \langle \pi_1,\ldots,\pi_k \rangle$ where:

- **$\pi_i \subseteq A_i \subseteq A$,**
- **π_i is applicable in state P_{i-1}, and**
- **the actions in π_i are independent.**

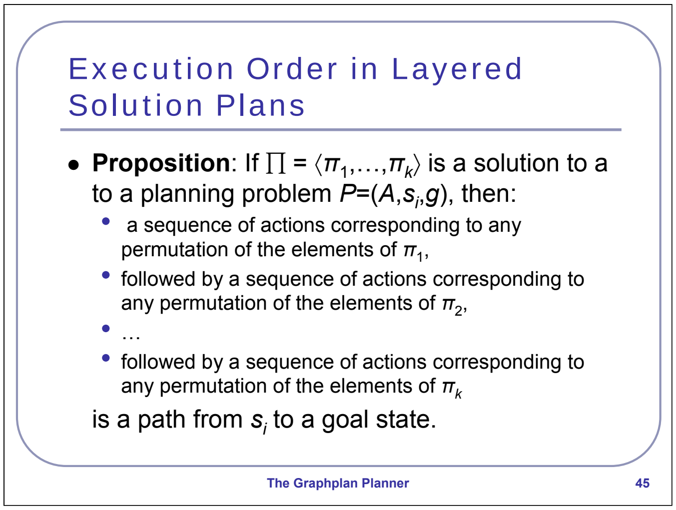


Layered Solution Plan

•A layered plan $\prod = \langle \pi_1,\ldots,\pi_k \rangle$ is a solution to a to a planning problem $P=(A,s_i,g)$ iff:

- **π_1 is applicable in s_i,**
- **for $j \in \{2\ldots k\}$, π_j is applicable in state $\gamma(\ldots\gamma(\gamma(s_i,\pi_1), \pi_2), \ldots \pi_{j-1})$, and**
- **$g \subseteq \gamma(\ldots\gamma(\gamma(s_i,\pi_1), \pi_2), \ldots, \pi_k)$.**

•note: independence of actions still not sufficient criterion for solution

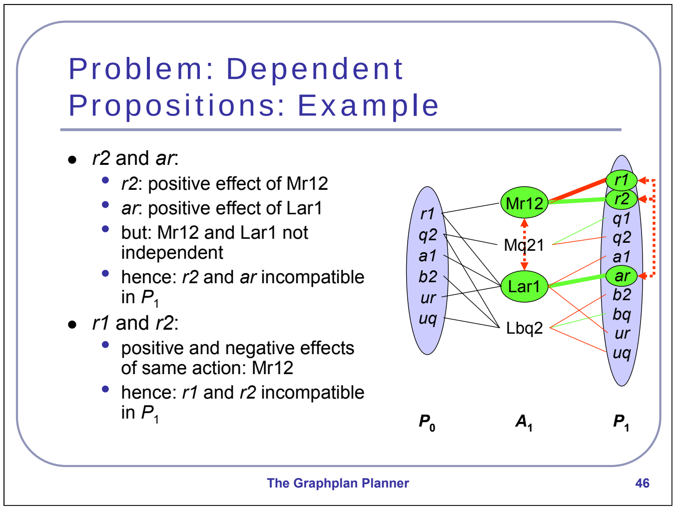


Execution Order in Layered Solution Plans

•**Proposition: If $\prod = \langle \pi_1,\ldots,\pi_k \rangle$ is a solution to a to a planning problem $P=(A,s_i,g)$, then:**

- **a sequence of actions corresponding to any permutation of the elements of π_1,**
- **followed by a sequence of actions corresponding to any permutation of the elements of π_2,**
- **…**
- **followed by a sequence of actions corresponding to any permutation of the elements of π_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**
  - •dependent actions cannot occur together same set of actions in a layered plan, e.g. in π_1
- •**hence: *r2* and *ar* incompatible in P_1**

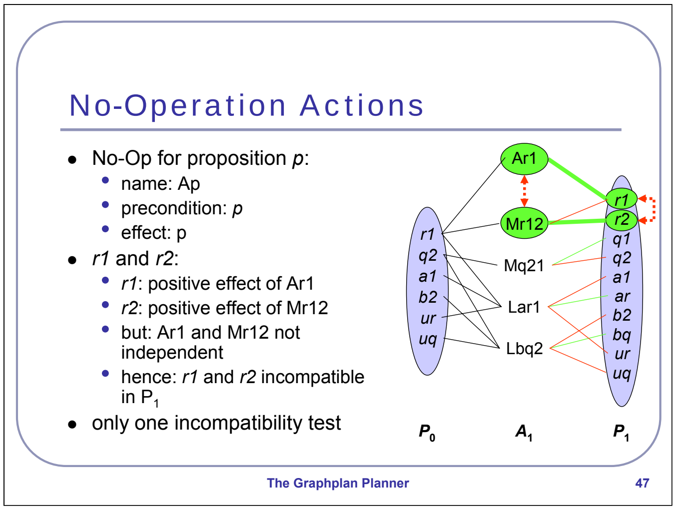
•***r1* and *r2***:

- •**positive and negative effects of same action: Mr12**
- •**hence: *r1* and *r2* incompatible in P_1**

•both cases: compatible if they are also

- •two positive effects of one action
- •the positive effects of two independent actions

•incompatible propositions: cannot be reached through preceding action layer (A_1)



No-Operation Actions

•**No-Op for proposition *p*:**

- •for every action layer and every proposition that may persist
- •**name: Ap**
- •**precondition: *p***
- •**effect: p**

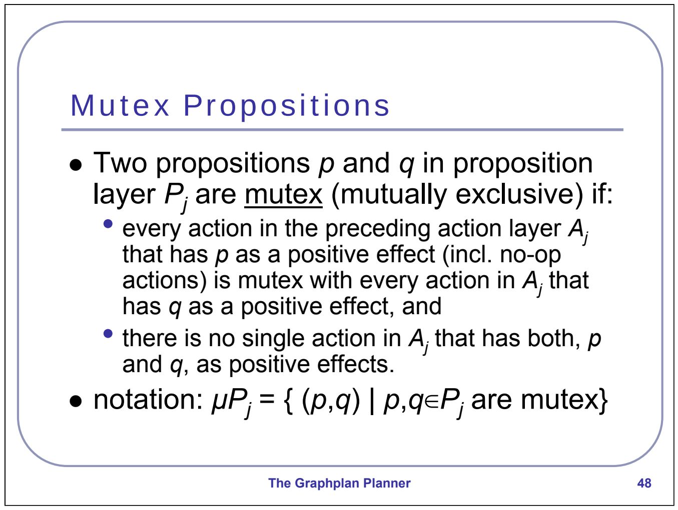
•***r1* and *r2*:**

- •***r1*: positive effect of Ar1**
- •***r2*: positive effect of Mr12**
- •**but: Ar1 and Mr12 not independent**
- •**hence: *r1* and *r2* incompatible in $\mathbf{P_1}$**

•**only one incompatibility test**

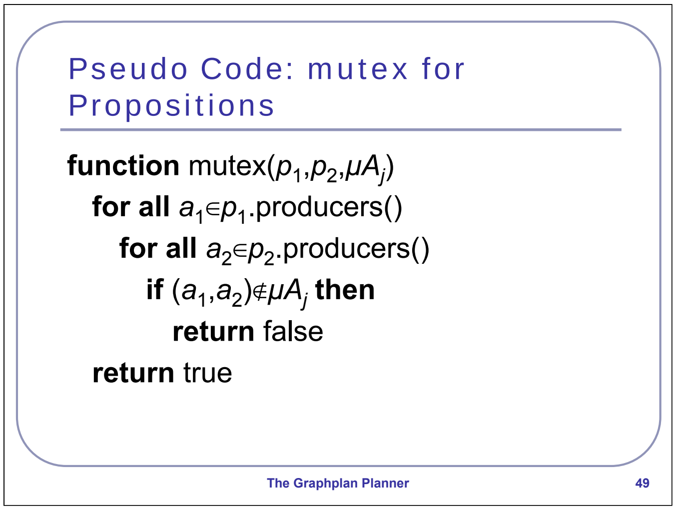
•previous slide: two types of incompatibility (positive effects of dependent actions + positive and negative effects of same action)

- •with no-ops: only first type needed (simplification)



Mutex Propositions

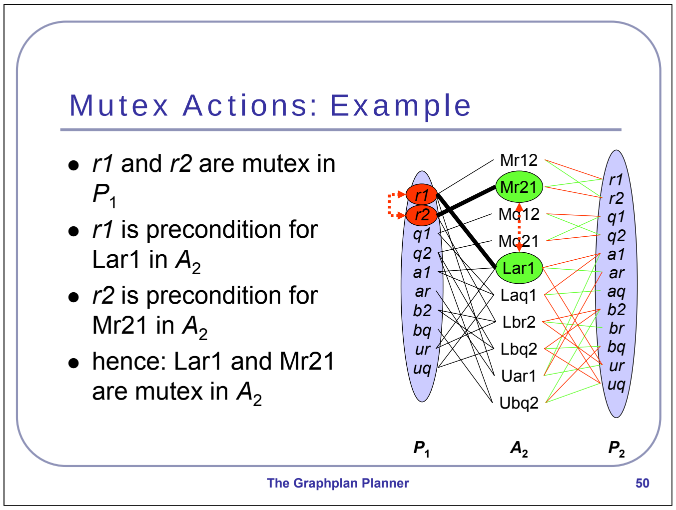
- **Two propositions p and q in proposition layer P_j are <u>mutex</u> (mutually exclusive) if:**
    - **every action in the preceding action layer A_j that has p as a positive effect (incl. no-op actions) is mutex with every action in A_j that has q as a positive effect, and**
    - need to define when two actions are mutex
        - obvious case: if they are dependent
    - **there is no single action in A_j that has both, p and q, as positive effects.**
- **notation: $\mu P_j = \{ (p,q) \mid p,q \in P_j \text{ are mutex}\}$**
- note: mutex relation for propositions is symmetrical (follows from definition)
- proposition layer P_1 contains 8 mutex pairs



Pseudo Code: mutex for Propositions

- **function mutex(p_1,p_2, μA_j)**
  - input: two propositions (from same layer), mutex relation between the actions in the preceding layer
- **for all $a_1 \in p_1$.producers()**
  - producers: actions in the preceding layer that have p_1 as a positive effect; should be stored with proposition node
- **for all $a_2 \in p_2$.producers()**
  - producers: see above
- **if (a_1,a_2)$\notin \mu A_j$ then**
  - test whether the action are in the given set of mutually exclusive actions
- **return false**
  - if not: consistent producers found; propositions are not mutex
- **return true**
  - no consistent producers found; propositions are mutex

- note: single action producing both is covered: action cannot be mutex with itself
- complexity: let m be number of actions in domain (incl. no-ops); $O(m^2)$



Mutex Actions: Example

•***r1*** **and** ***r2*** **are mutex in** $\boldsymbol{P_1}$

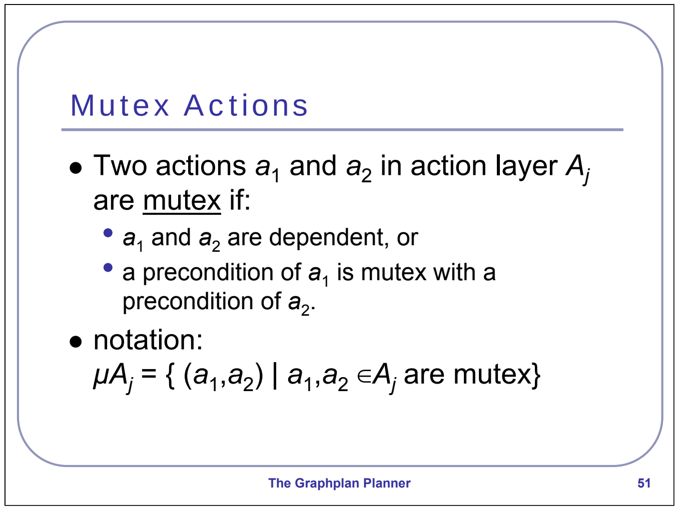
•***r1*** **is precondition for Lar1 in** $\boldsymbol{A_2}$

•***r2*** **is precondition for Mr21 in** $\boldsymbol{A_2}$

•**hence: Lar1 and Mr21 are mutex in** $\boldsymbol{A_2}$

•dependency between actions in action layer A_j leads to mutex between propositions in P_j

•mutex between propositions in P_j leads to mutex between actions in action layer A_{j+1}



Mutex Actions

- **Two actions a_1 and a_2 in action layer A_j are <u>mutex</u> if:**
  - **a_1 and a_2 are dependent, or**
    - dependent actions are necessarily mutex
  - **a precondition of a_1 is mutex with a precondition of a_2.**
  - dependency is domain-specific, i.e. not problem-specific
  - mutex-relation is problem specific
    - pair of actions/propositions may be mutex in one layer but not so in another
- **notation:**
  $\mu A_j = \{ (a_1,a_2) \mid a_1,a_2 \in A_j \text{ are mutex}\}$
- action layer A_1 contains 2 mutex (dependent) pairs
- action layer A_2 contains 24 mutex pairs (not all dependent)
- note: mutex relation (for actions and propositions) is symmetrical (follows from definition)

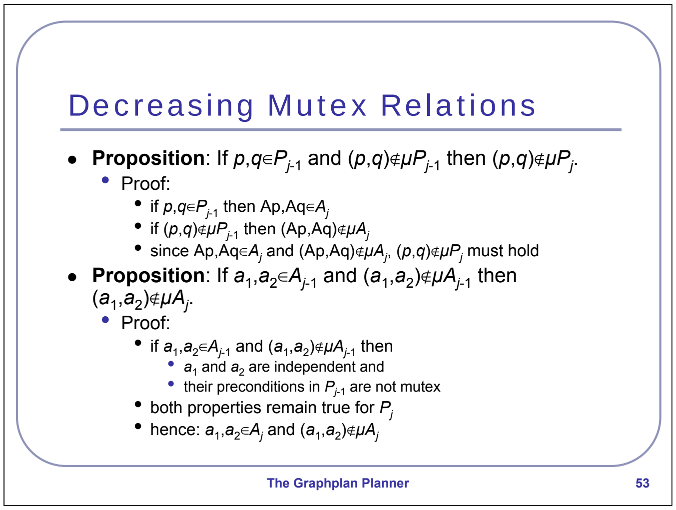
Pseudo Code: mutex for Actions

function mutex(a_1,a_2,μP)
  if ¬independent(a_1,a_2) **then**
    return true
  for all $p_1 \in$precond(a_1)
    for all $p_2 \in$precond(a_2)
      if (p_1,p_2)$\in \mu P$ **then return** true
  return false

The Graphplan Planner 52

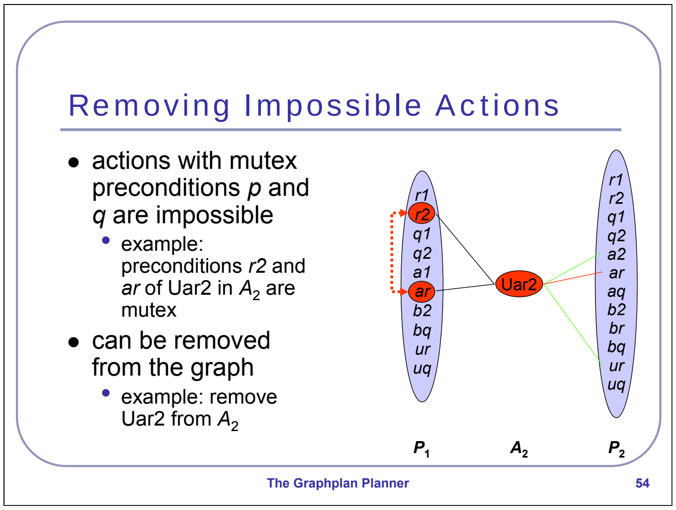
Pseudo Code: mutex for Actions

- **function mutex(a_1,a_2,μP)**
  - μP – mutex relations from the preceding proposition layer
- **if ¬independant(a_1,a_2) then**
- **return true**
- **for all $p_1 \in$precond(a_1)**
- **for all $p_2 \in$precond(a_2)**
- **if (p_1,p_2)$\in \mu P$ then return true**
- **return false**
- complexity: let b = max number preconditions/pos. effects/neg effects: $O(b^2)$



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)$\notin \mu A_j$, **$(p,q) \notin \mu P_j$ 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 $(a_1,a_2) \notin \mu A_j$**
- mutex relations are monotonically decreasing (between layers with the same propositions)



Removing Impossible Actions

•**actions with mutex preconditions *p* and *q* are impossible**

  •**example: preconditions *r2* and *ar* of Uar2 in A_2 are mutex**

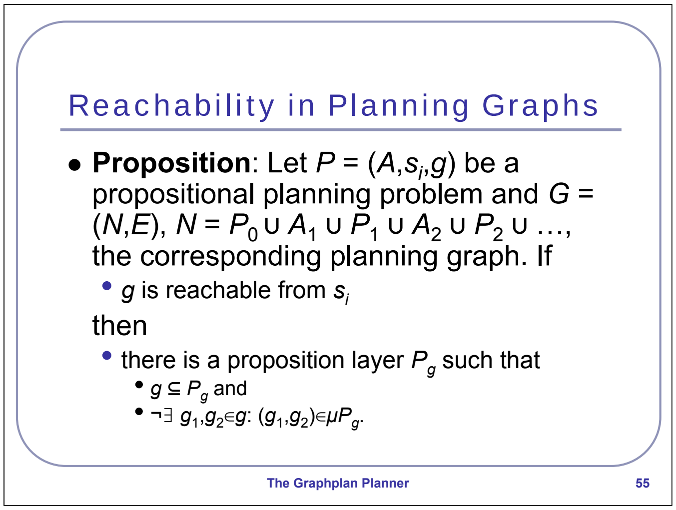
•action with mutex preconditions can never be part of any layered plan (will violate applicability condition in definition)

•**can be removed from the graph**

  •**example: remove Uar2 from A_2**

•mutex pair of actions must remain in graph because one of the actions may be used in final plan

•note: still consistent with monotonically increasing actions



Reachability in Planning Graphs

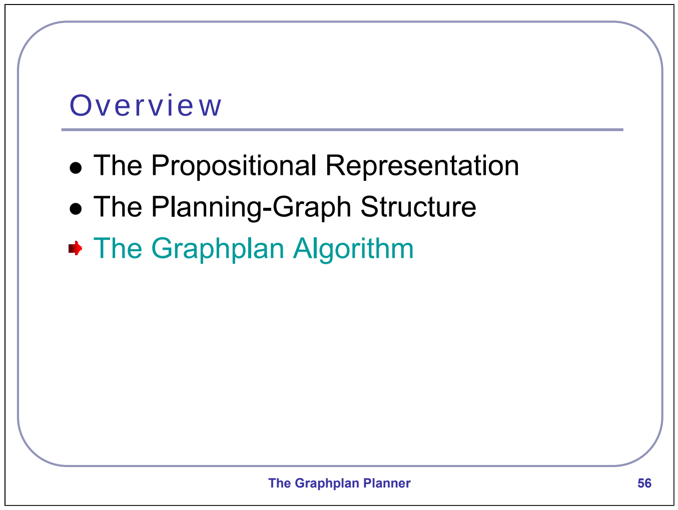
•Proposition: Let $P = (A,s_i,g)$ be a propositional planning problem and $G = (N,E)$, $N = P_0 \cup A_1 \cup P_1 \cup A_2 \cup P_2 \cup \ldots$, the corresponding planning graph. If

- **g is reachable from s_i**

• then

- **there is a proposition layer P_g such that**
  - **$g \subseteq P_g$ and**
  - **$\neg\exists\ g_1,g_2 \in g: (g_1,g_2) \in \mu P_g$.**

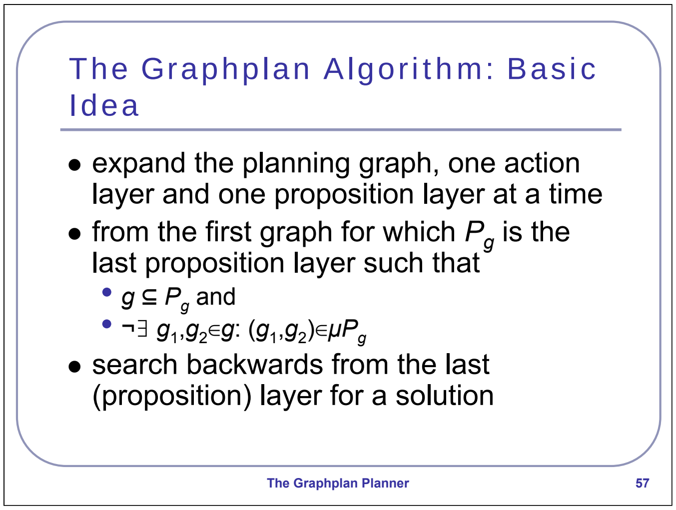
•still only necessary condition, but relatively efficient to compute



Overview

The Propositional Representation

- **The Planning-Graph Structure**
  - just done: defining a new graph that is more efficient to generate and a necessary criterion for solution containment
- **The Graphplan Algorithm**
  - now: an algorithm for searching the planning graph for a solution plan



The Graphplan Algorithm: Basic Idea

•expand the planning graph, one action layer and one proposition layer at a time

  •similar to iterative deepening: discover new part of the search space with each iteration

•from the first graph for which P_g is the last proposition layer such that

  •$g \subseteq P_g$ **and**

  •$\neg\exists\ g_1,g_2 \in g: (g_1,g_2) \in \mu P_g$

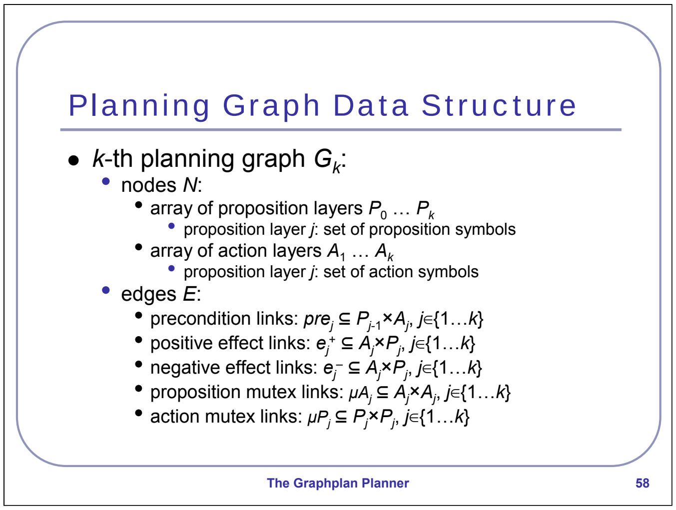
  •no need to search for solutions in graph with fewer layers; see last proposition

•search backwards from the last (proposition) layer for a solution

•two major steps:

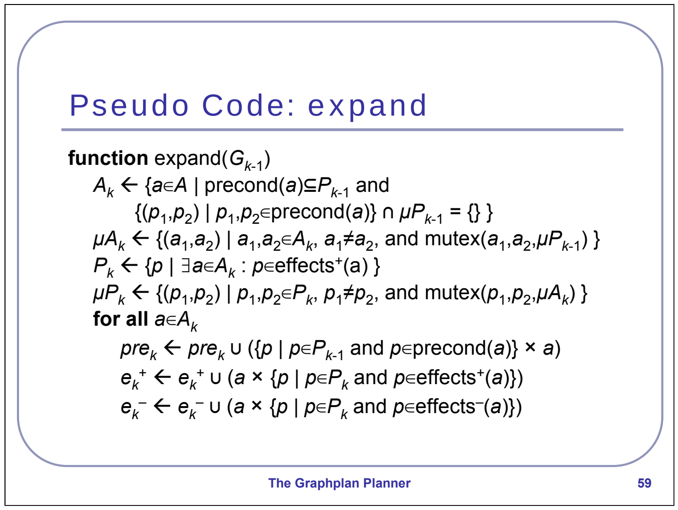
  •expansion of planning graph to next proposition layer

  •searching a given planning graph for a solution



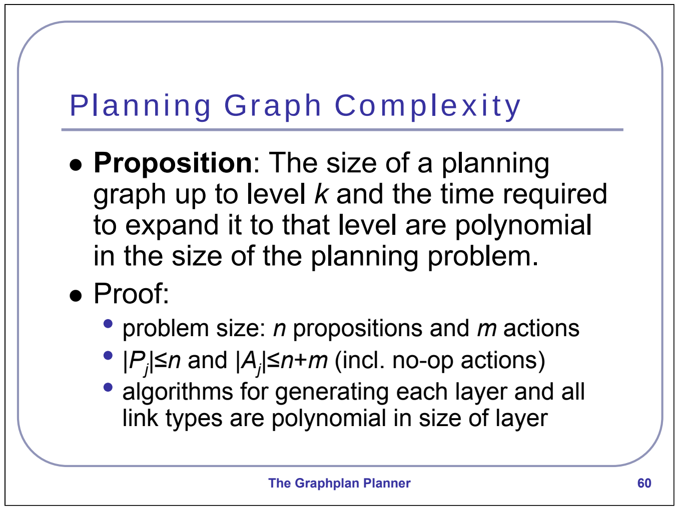
Planning Graph Data Structure

- **k-th planning graph G_k:**
  - **nodes N:**
    - **array of proposition layers $P_0 \dots P_k$**
      - **proposition layer j: set of proposition symbols**
    - **array of action layers $A_1 \dots A_k$**
      - **proposition layer j: set of action symbols**
  - **edges E:**
    - **precondition links: $pre_j \subseteq P_{j-1} \times A_j$, $j \in \{1 \dots k\}$**
    - **positive effect links: $e_j^+ \subseteq A_j \times P_j$, $j \in \{1 \dots k\}$**
    - **negative effect links: $e_j^- \subseteq A_j \times P_j$, $j \in \{1 \dots k\}$**
    - **proposition mutex links: $\mu A_j \subseteq A_j \times A_j$, $j \in \{1 \dots k\}$**
    - **action mutex links: $\mu P_j \subseteq P_j \times P_j$, $j \in \{1 \dots k\}$**
- note: instance of this data structure does not depend on problem
- initial planning graph: $P_0 = s_i$; rest is empty sets



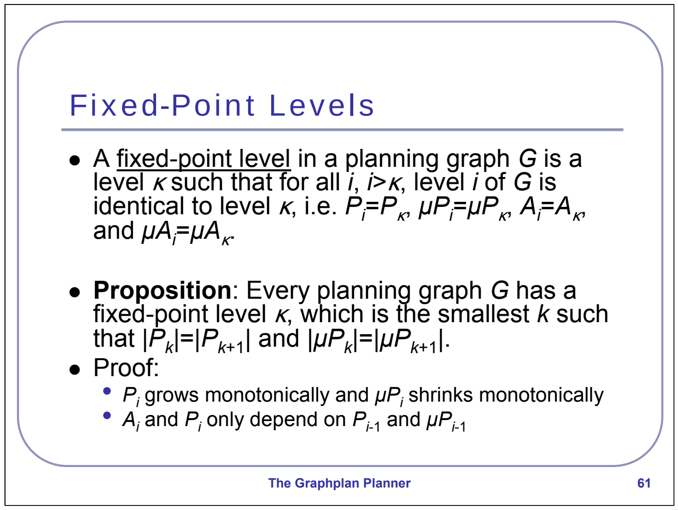
Pseudo Code: expand

- **function expand($G_{k\text{-}1}$)**
- $A_k \leftarrow \{a \in A \mid \text{precond}(a) \subseteq P_{k\text{-}1} \text{ and } \{(p_1,p_2) \mid p_1,p_2 \in \text{precond}(a)\} \cap \mu P_{k\text{-}1} = \{\}\ \}$
    - actions with satisfied, non-mutex preconditions (incl. no-ops)
- $\mu A_k \leftarrow \{(a_1,a_2) \mid a_1,a_2 \in A_k,\ a_1 \neq a_2,\ \text{and mutex}(a_1,a_2,\mu P_{k\text{-}1})\ \}$
- $P_k \leftarrow \{p \mid \exists a \in A_k : p \in \text{effects}^+(\text{a})\ \}$
    - union of all positive effects
- $\mu P_k \leftarrow \{(p_1,p_2) \mid p_1,p_2 \in P_k,\ p_1 \neq p_2,\ \text{and mutex}(p_1,p_2,\mu A_k)\ \}$
- **for all** $a \in A_k$
- $pre_k \leftarrow pre_k \cup (\{p \mid p \in P_{k\text{-}1} \text{ and } p \in \text{precond}(a)\} \times a)$
- $e_k^+ \leftarrow e_k^+ \cup (a \times \{p \mid p \in P_k \text{ and } p \in \text{effects}^+(a)\})$
- $e_k^- \leftarrow e_k^- \cup (a \times \{p \mid p \in P_k \text{ and } p \in \text{effects}^-(a)\})$



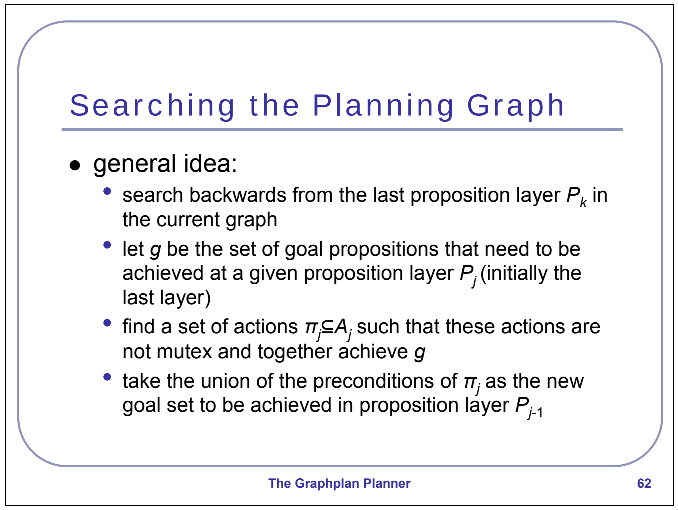
Planning Graph Complexity

- **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**
  - **$|P_j| \leq n$ and $|A_j| \leq n+m$ (incl. no-op actions)**
  - **algorithms for generating each layer and all link types are polynomial in size of layer**



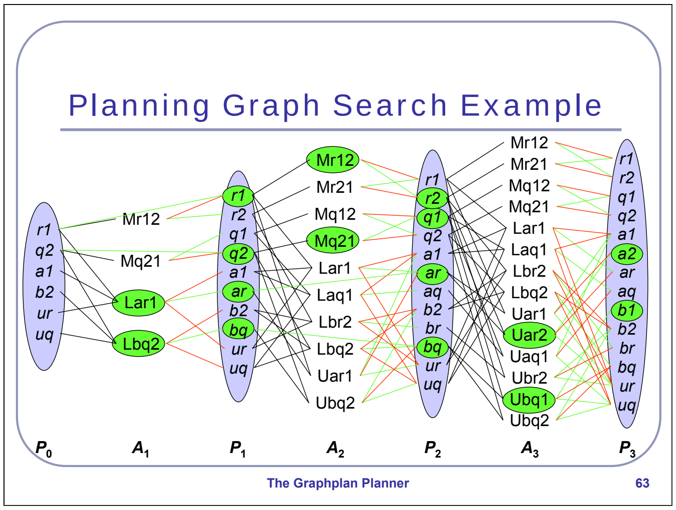
Fixed-Point Levels

- **A <u>fixed-point level</u> in a planning graph *G* is a level κ such that for all *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 $|P_k|=|P_{k+1}|$ and $|\mu P_k|=|\mu P_{k+1}|$.**
  - $|P_k|=|P_{k+1}|$ implies $P_k=P_{k+1}$
- **Proof:**
  - **P_i grows monotonically and μP_i shrinks monotonically**
    - μP_i shrinks monotonically: for equal P_i
  - **A_i and P_i only depend on P_{i-1} and μP_{i-1}**
- time complexity: $O(n+m)$ from fixed point level; only copying required



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 π_j as the new goal set to be achieved in proposition layer P_{j-1}



Planning Graph Search Example

•initial goal: *a2* and *b1*

•only one incoming positive effect link per goal (but no-ops not shown)

•achievable with Uar2 and Ubq1 (which are not mutex; mutex relations not shown)

•precondition links indicate sub-goal at next layer

•new sub-goal at P_2: *r2*, *q1*, *ar*, *bq*

•only one incoming positive effect link per goal condition (but no-ops not shown)

- •achieve *ar* and *bq* with no-ops
- •achieve *r2* with Mr12 and *q1* with Mq21

•precondition links (for Mr12 and Mq21) indicate some sub-goal at next layer

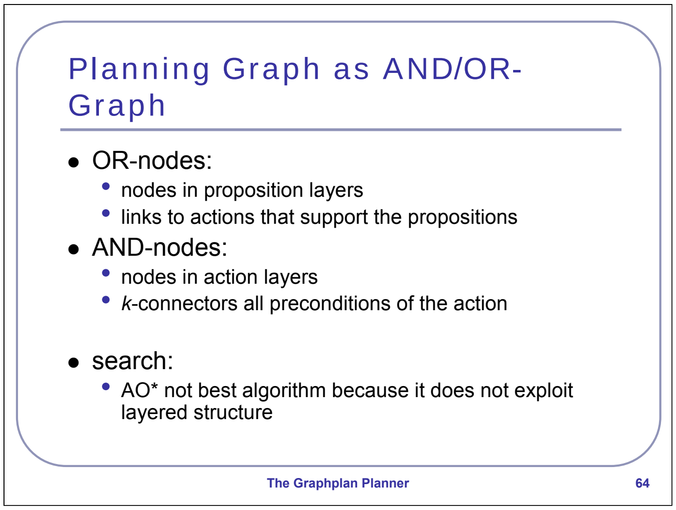
•complete sub-goal (incl. preconditions of no-ops) at P_1: *r1*, *q2*, *ar*, *bq*

•only one incoming positive effect link per goal condition (but no-ops not shown)

- •achieve *r1* and *q2* with no-ops
- •achieve *ar* with Lar1 and *bq* with Lbq2

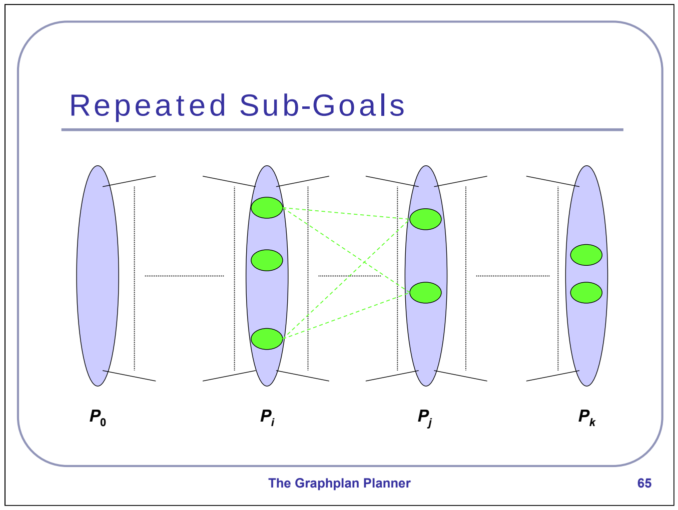
•precondition links (for Lar1 and Lbq2) indicate some sub-goal at next layer

•complete sub-goal (incl. preconditions of no-ops) at P_0: complete initial state



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

•ultimate goal leads to possible sub-goals at P_j

•possible sub-goals at P_j lead to possible sub-goals at P_i

  •search to initial proposition layer to see whether sub-goals can be achieved

  •suppose: sub-goals at P_i cannot be achieved

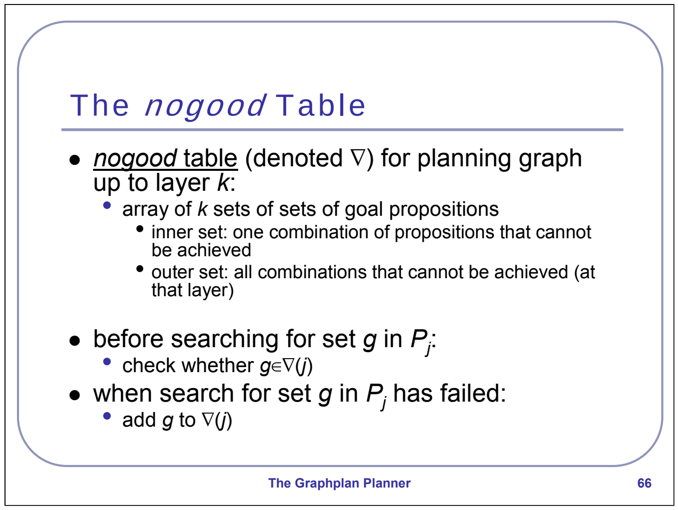
•backtrack to later layer, say P_j

•possible sub-goals at P_j may lead to same possible sub-goals at P_i, but in a different way

  •no need to repeat search: same sub-goals at same layer still cannot be achieved

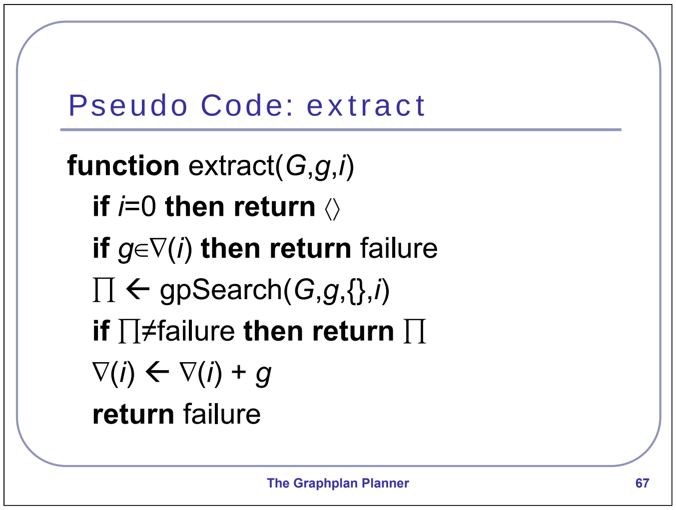
  •generalization: same some sub-goals at same or earlier layer still cannot be achieved

    •otherwise no-op would achieve sub-goal at later layer



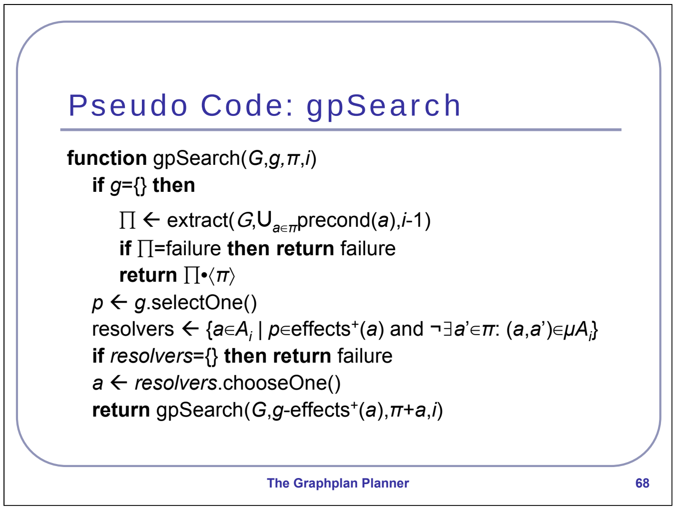
The *nogood* Table

- **<u>*nogood* table</u> (denoted ∇) 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)**
  - mutex only gives pairs of propositions that cannot be achieved together, *nogood* table gives impossible tuples
- **before searching for set *g* in P_j:**
  - **check whether $g \in \nabla(j)$**
  - actually: in *j* or later layer
- **when search for set *g* in P_j has failed:**
  - **add *g* to $\nabla(j)$**
  - or move?



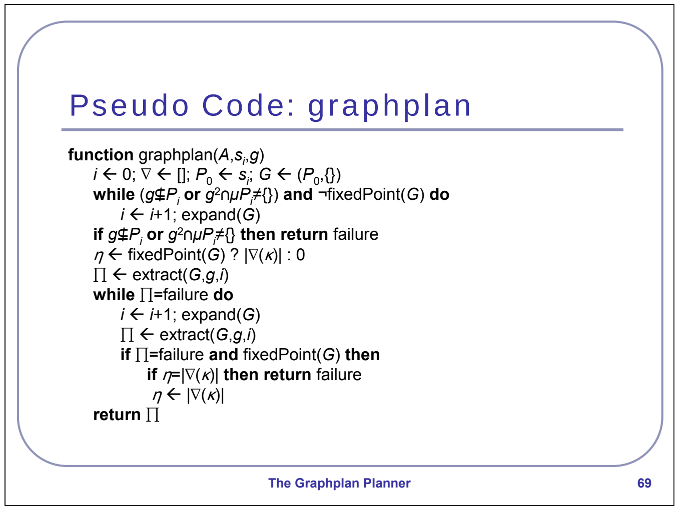
Pseudo Code: extract

- **function extract(*G*,*g*,*i*)**
    - inputs: planning graph *G*, set of propositions (sub-goals) *g*, and layer at which sub-goals need to be achieved *i*
    - output: a layered plan $\langle \pi_1,\ldots,\pi_i \rangle$ that achieves *g* at *i* in *G* or failure if there is no such plan
- **if *i*=0 then return ⟨⟩**
    - trivial success with empty plan
- **if $g \in \nabla(i)$ then return failure**
    - sub-goals have resulted in failure before
- **π_i ← gpSearch(*G*,*g*,{},*i*)**
    - perform the search
- **if $\pi_i \neq$ failure then return π_i**
    - the search was successful
- **$\nabla(i) \leftarrow \nabla(i) + g$**
    - unsuccessful search: remember unachievable sub-goals
- **return failure**



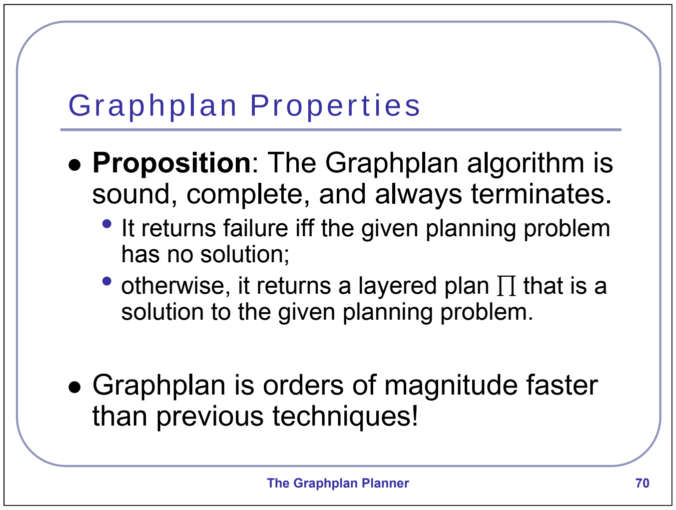
Pseudo Code: gpSearch

- **function gpSearch(G,g,π,i)**
  - inputs: planning graph G, remaining sub-goals g, and set of actions already committed to π, both at level i
  - outputs: layered plan
- **if g={} then**
  - all actions chosen
- $\prod \leftarrow$ extract(G,$\cup_{a\in\pi}$**precond(a),i-1)**
- **if $\prod$=failure then return failure**
- **return $\prod \bullet \langle\pi\rangle$**
- **$p \leftarrow g$.selectOne()**
  - no need to backtrack here; order only important for efficiency
- **resolvers $\leftarrow \{a\in A_i \mid p\in \text{effects}^+(a)$ and $\neg\exists a'\in\pi: (a,a')\in\mu A_i\}$**
- **if *resolvers*={} then return failure**
- **$a \leftarrow$ *resolvers*.chooseOne()**
  - non-deterministic choice point; backtrack to here
- **return GPSearch(G,g-effects$^+$(a),π+a,i)**



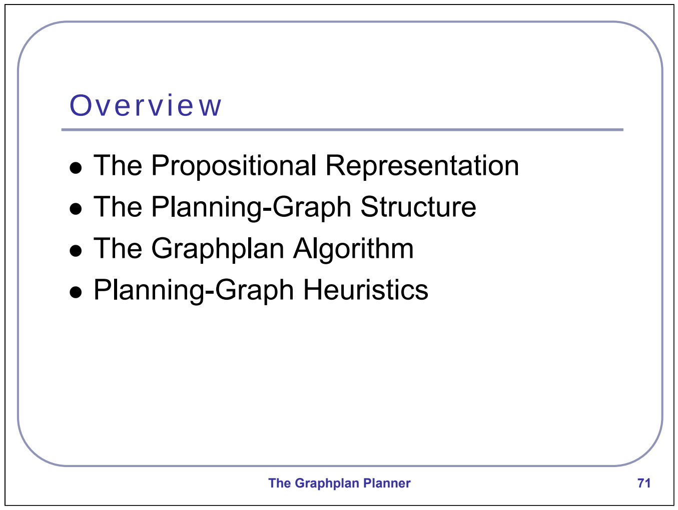
Pseudo Code: graphplan

- **function graphplan(A,s_i,g)**
  - given planning problem, return layered solution plan
- **$i \leftarrow 0$; $\nabla \leftarrow []$; $P_0 \leftarrow s_i$; $G \leftarrow (P_0,\{\})$**
- **while ($g \nsubseteq P_i$ or $g^2 \cap \mu P_i \neq \{\}$) and $\neg$fixedPoint(G) do**
- **$i \leftarrow i+1$; expand(G)**
  - planning graph expanded until solution possible or fixed point reached
- **if $g \nsubseteq P_i$ or $g^2 \cap \mu P_i \neq \{\}$ then return failure**
  - test necessary criterion
- **$\eta \leftarrow$ fixedPoint(G) ? $|\nabla(\kappa)|$: 0**
  - used to test when expansion will not work
- **$\prod \leftarrow$ extract(G,g,i)**
- **while $\prod$=failure do**
- **$i \leftarrow i+1$; expand(G)**
- **$\prod \leftarrow$ extract(G,g,i)**
- **if $\prod$=failure and fixedPoint(G) then**
- **if $\eta = |\nabla(\kappa)|$ then return failure**
- **$\eta \leftarrow |\nabla(\kappa)|$**
- **return $\prod$**



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** $\prod$ that is a solution to the given planning problem.
- Graphplan is orders of magnitude faster than previous techniques!
    - caveat: restriction to propositional STRIPS



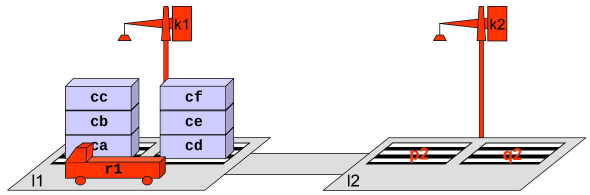
Overview

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

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 $p \in g$: min-layer(p) = index of first proposition layer in planning graph that contains p
  - admissible heuristic: max($p \in g$): min-layer(p)
  - not admissible: sum($p \in g$): 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

Overview

➡The Propositional Representation

•The Planning-Graph Structure

•The Graphplan Algorithm