Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili

AI Planning

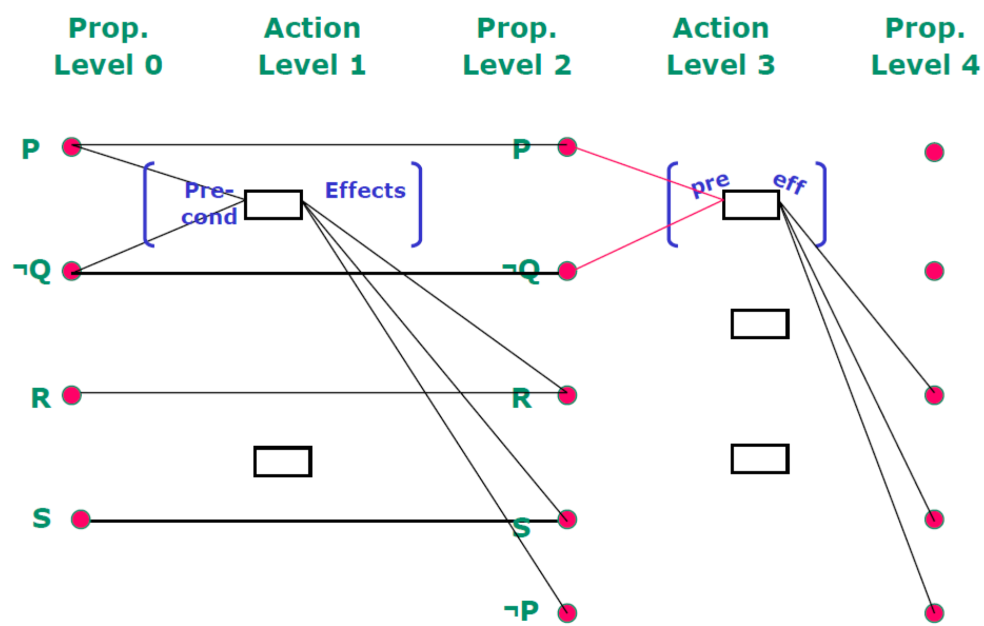
Hatem A. Rashwan

Graphplan and Advanced Heuristics

Graphplan

Graph Plan

- A propositional planner, that is, there are no variables
  - assertions Simpler – don't have to worry about matching
  - Bigger – if you have six blocks, you need 36 propositions to represent all On(x,y)



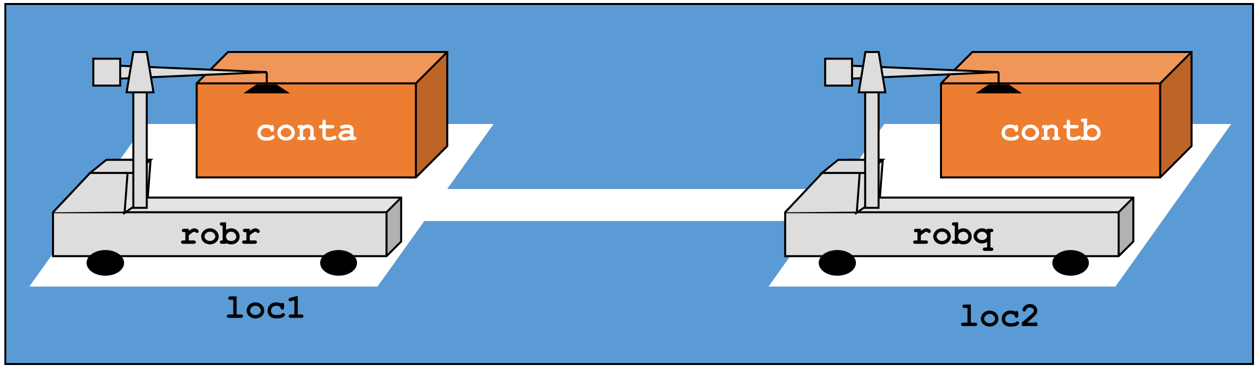
1. Make a plan graph of depth k
2. Search for a solution
3. If succeed, return a plan
4. Else k=k+1
5. Go to 1.

GraphPlan centres work on a data structure called a plan graph. A plan graph looks like this in the figure. You have a bunch of levels. You start with level zero, level one, level two.

Overview

- **A Propositional DWR Example**
- The Basic Planning Graph (No Mutex)
- Layered Plans
- Mutex Propositions and Actions
- Graphplan properties

Example: Simplified DWR Problem



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

Simplified DWR Problem: STRIPS Operators

- **move(*r*,*l*,*l'*)**
  - precond: at(*r*,*l*), adjacent(*l*,*l'*)
  - effects: at(*r*,*l'*), ¬at(r,*l*)
- **load(*c*,*r*,*l*)**
  - precond: at(*r*,*l*), in(*c*,*l*), unloaded(*r*)
  - effects: loaded(*r*,*c*), ¬in(*c*,*l*), ¬unloaded(*r*)
- **unload(*c*,*r*,*l*)**
  - precond: at(*r*,*l*), loaded(*r*,*c*)
  - effects: unloaded(*r*), in(*c*,*l*), ¬loaded(*r*,*c*)

Simplified DWR Problem: State Proposition Symbols

- robots:
  - *r1* and *r2*: 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)

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

Simplified DWR Problem: Propositions 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, Lbr1, Lbr2, Lbq1, and Lbq2 correspondingly
- unload actions:
  - Uar1: unload(conta,robr,loc1); Uar2, Uaq1, Uaq2, Ubr1, Ubr2, Ubq1, and Ubq2 correspondingly

Overview

- A Propositional DWR Example
- **The Basic Planning Graph (No Mutex)**
- Layered Plans
- Mutex Propositions and Actions
- Graphplan properties

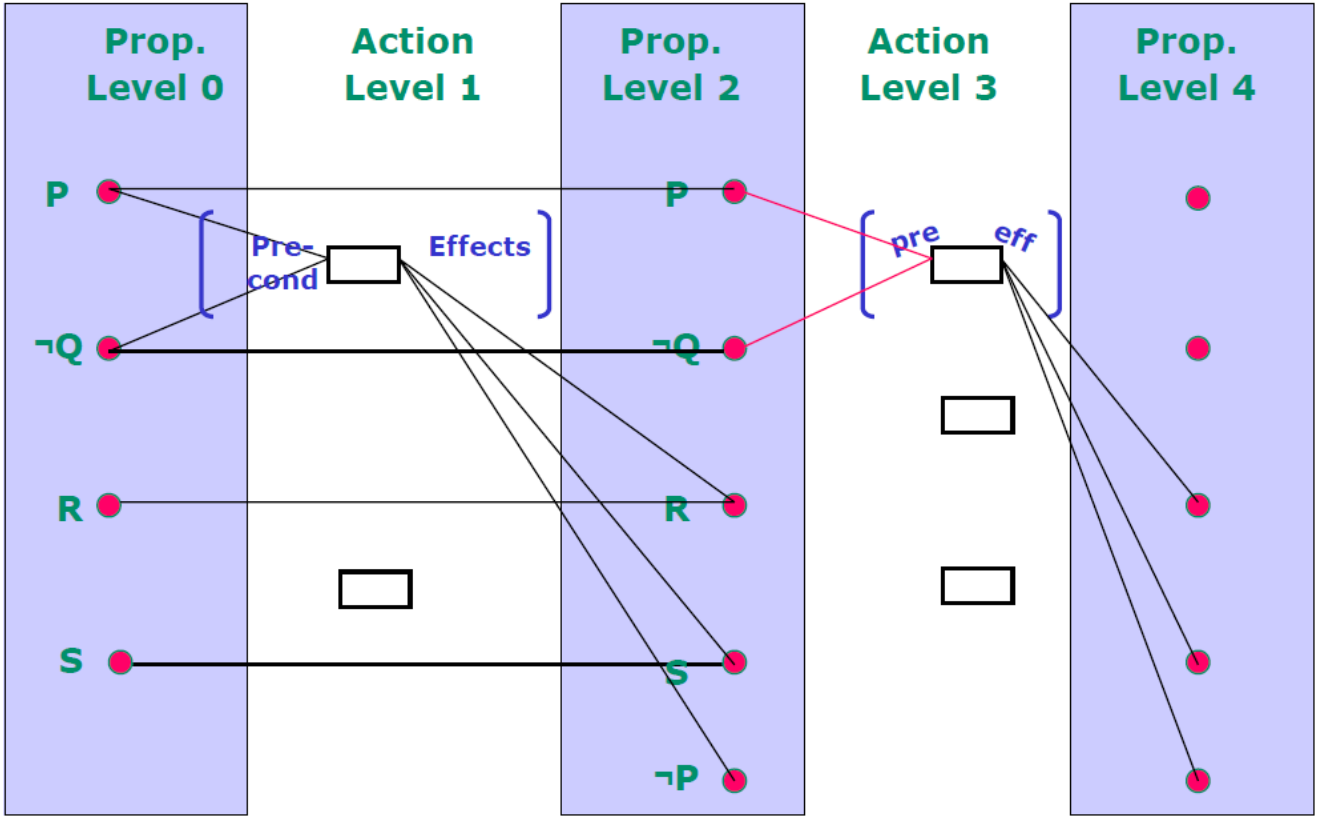
Planning Graph: Nodes

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

Planning Graph: Edges

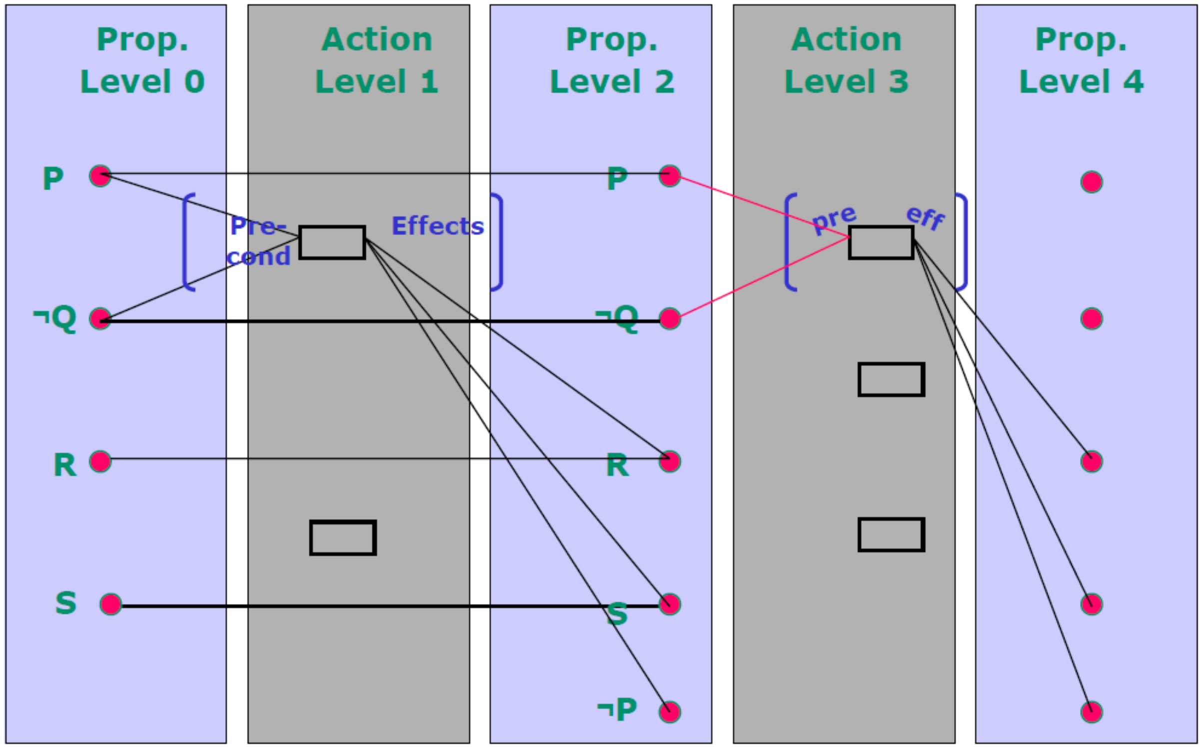
- from proposition $p \in P_{j\text{-}1}$ to action $a \in A_j$:
  - if: $p \in \mathrm{precond}(a)$
- from action $a \in A_j$ to layer $p \in P_j$:
  - positive arc if: $p \in \mathrm{effects}^{+}(a)$
  - negative arc if: $p \in \mathrm{effects}^{-}(a)$

Graph Plan Example



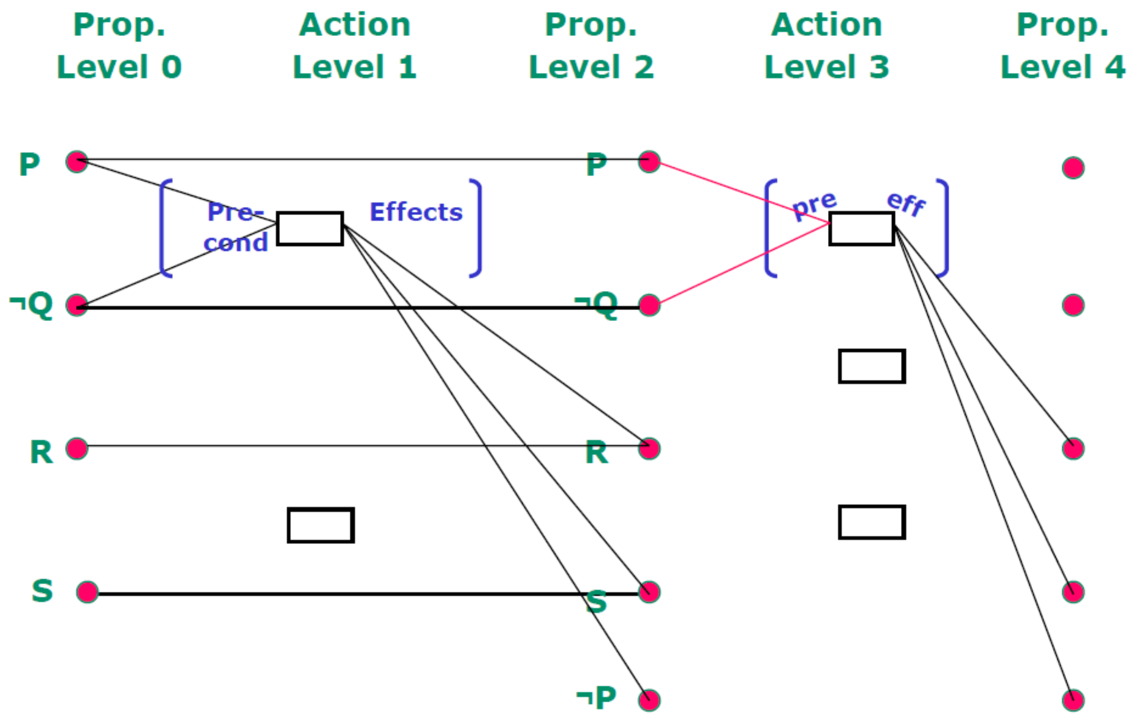
At the even- numbered levels you have propositions, which they draw as a little dot.

Graph Plan Example



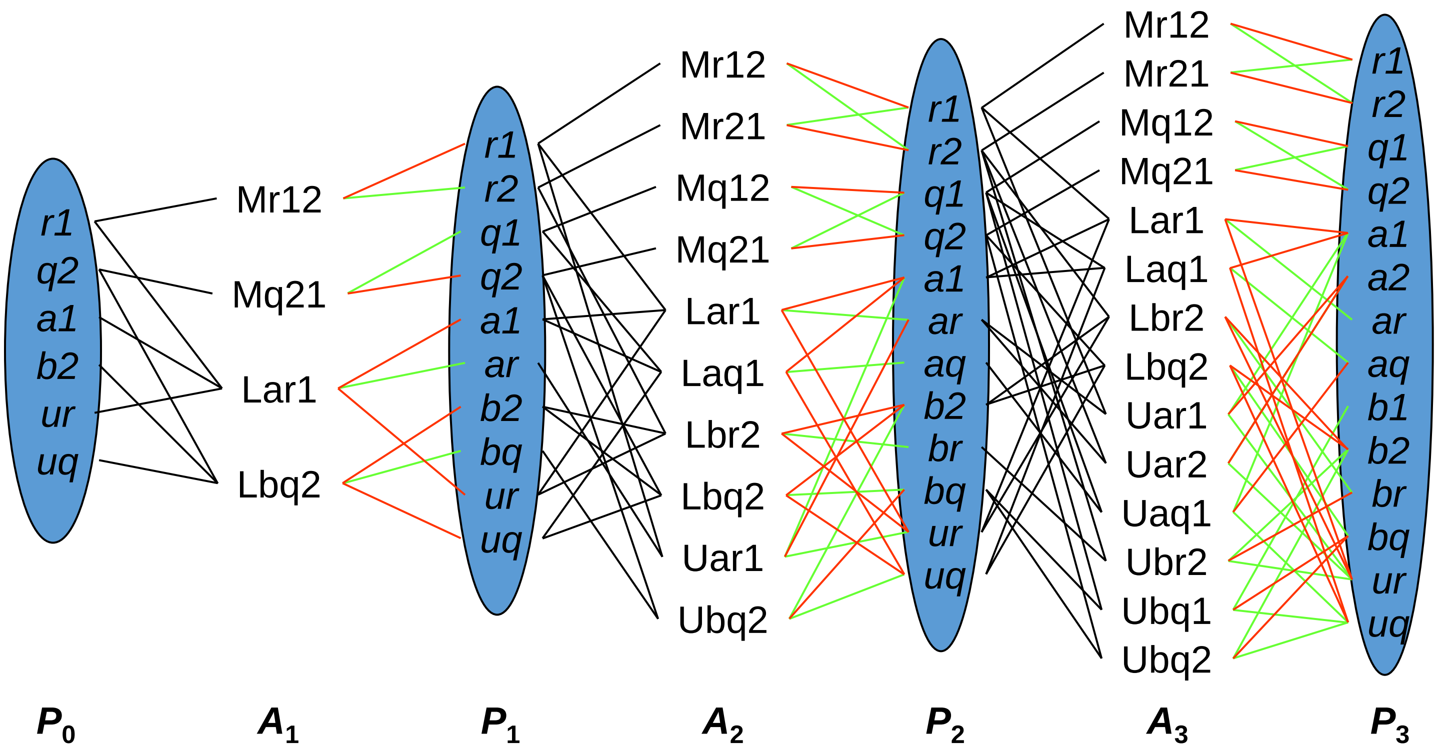
- Three proposition levels (levels 0, 2, and 4) and two action levels (levels 1 and 3).
- To encode depth-two plans (because there are two layers of actions).
  - Action level 1 has the actions that we might choose to do on the first step,
  - Action level 3 has the actions we might choose to do on the second step.

Graph Plan Example



- Start by making a graph with levels 0 through 2, corresponding to a depth 1 plan,
- Search for a satisfactory plan within that graph. If we can't find one,
- Extend the graph out by two more layers (an action layer and a proposition layer),
- Then find a depth 2 plan.

Planning Graph Example



The goal is to swap the containers: {a2, q1}

Reachability in the Planning Graph

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

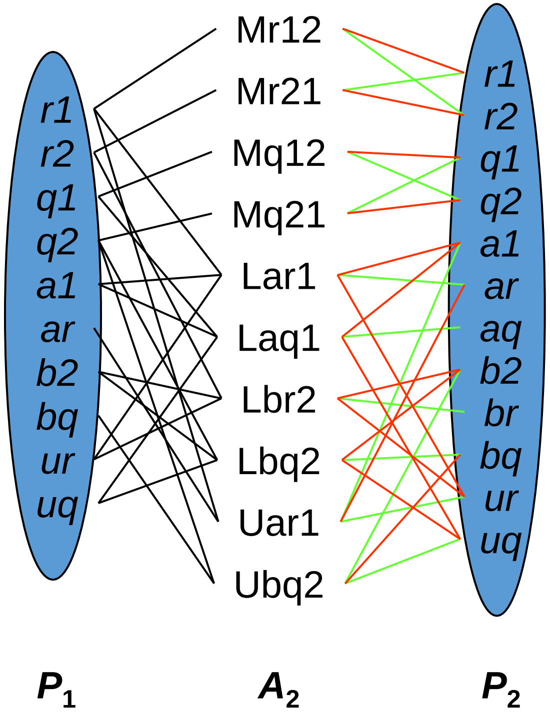
- necessary condition, but not sufficient
- low complexity:
  - planning graph is of polynomial size and
  - can be computed in polynomial time

Overview

- A Propositional DWR Example
- The Basic Planning Graph (No Mutex)
- **Layered Plans**
- Mutex Propositions and Actions
- Graphplan properties

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

- Two actions a_1 and a_2 are <u>independent</u> iff:
  - $\text{effects}^-(a_1) \cap (\text{precond}(a_2) \cup \text{effects}^+(a_2)) = \{\}$ and
  - $\text{effects}^-(a_2) \cap (\text{precond}(a_1) \cup \text{effects}^+(a_1)) = \{\}$.

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

- The final solution Π is {π1, π2, π3,…., πk}

Pseudo Code: independent

function independent(a_1,a_2)

  for all $p \in \text{effects}^-(a_1)$

    if $p \in \text{precond}(a_2)$ **or** $p \in \text{effects}^+(a_2)$ **then**

      return false

  for all $p \in \text{effects}^-(a_2)$

    if $p \in \text{precond}(a_1)$ **or** $p \in \text{effects}^+(a_1)$ **then**

      return false

  return true

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

Layered Solution Plan

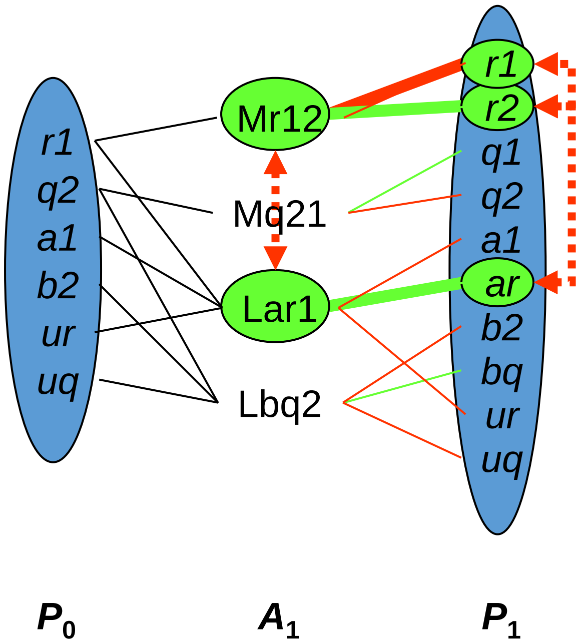
- A layered plan $\prod = \langle \pi_1,\ldots,\pi_k \rangle$ is a solution to a to a planning problem $P=(A,s_i,g)$ iff:
  - π_1 is applicable in s_i,
  - for $j \in \{2\ldots k\}$, π_j is applicable in state $\gamma(\ldots\gamma(\gamma(s_i,\pi_1), \pi_2), \ldots \pi_{j-1})$, and
  - $g \subseteq \gamma(\ldots\gamma(\gamma(s_i,\pi_1), \pi_2), \ldots, \pi_k)$.

Overview

- A Propositional DWR Example
- The Basic Planning Graph (No Mutex)
- Layered Plans
- **Mutex Propositions and Actions**
- Graphplan properties

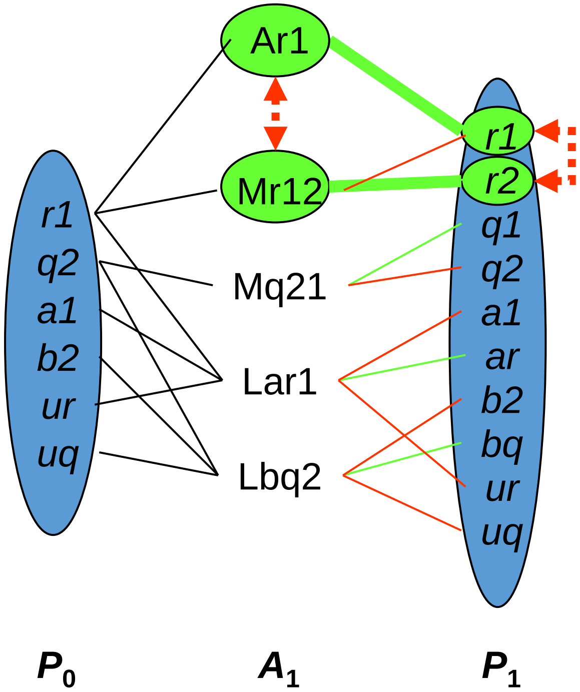
Problem: Dependent Propositions: Example

- *r2* and *ar*:
  - *r2*: positive effect of Mr12
  - *ar*: positive effect of Lar1
  - but: Mr12 and Lar1 not independent
  - hence: *r2* and *ar* incompatible in P_1
- *r1* and *r2*:
  - positive and negative effects of same action: Mr12
  - hence: *r1* and *r2* incompatible in P_1



No-Operation Actions

- No-Op for proposition *p*:
  - name: Ap
  - precondition: *p*
  - effect: p
- *r1* and *r2*:
  - *r1*: positive effect of Ar1
  - *r2*: positive effect of Mr12
  - but: Ar1 and Mr12 not independent
  - hence: *r1* and *r2* incompatible in P_1



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
  - there is no single action in A_j that has both, p and q, as positive effects.

- notation: $\mu P_j = \{ (p,q) \mid p,q \in P_j \text{ are mutex}\}$

Pseudo Code: mutex for Propositions

function mutex(p_1,p_2,μA_j)
  for all $a_1 \in p_1$.producers()
    for all $a_2 \in p_2$.producers()
      if $(a_1,a_2) \notin \mu A_j$ **then**
        return false
  return true

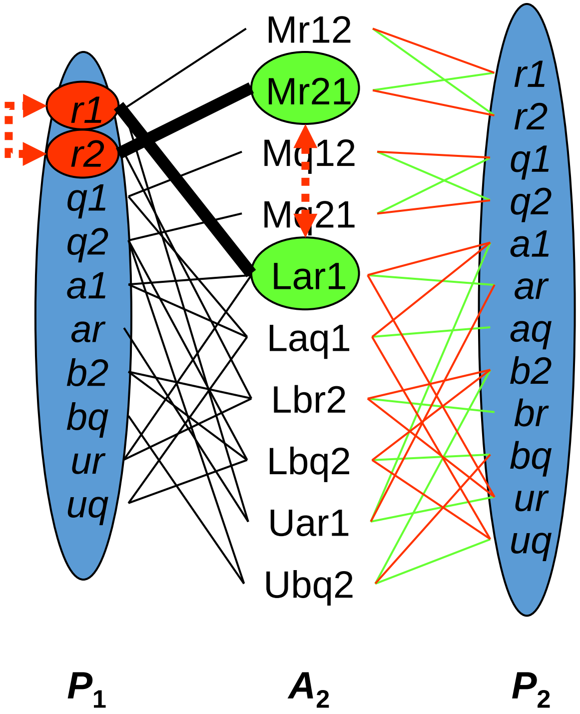
Mutex Actions

- 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
  - a precondition of a_1 is mutex with a precondition of a_2.

- notation: $\mu A_j = \{ (a_1,a_2) \mid a_1,a_2 \in A_j$ are mutex$\}$

Mutex Actions: Example

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



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

Overview

- A Propositional DWR Example
- The Basic Planning Graph (No Mutex)
- Layered Plans
- Mutex Propositions and Actions
- **Graphplan properties**

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 ∏ that is a solution to the given planning problem.

- Graphplan is orders of magnitude faster than previous techniques!

Advanced Heuristics

Overview

- **Simple Planning Graph Heuristics**
- The FF Planner

Forward State-Space Search with A*

- A* is optimally efficient: For a given heuristic function, no other algorithm is guaranteed to expand fewer nodes than A*.

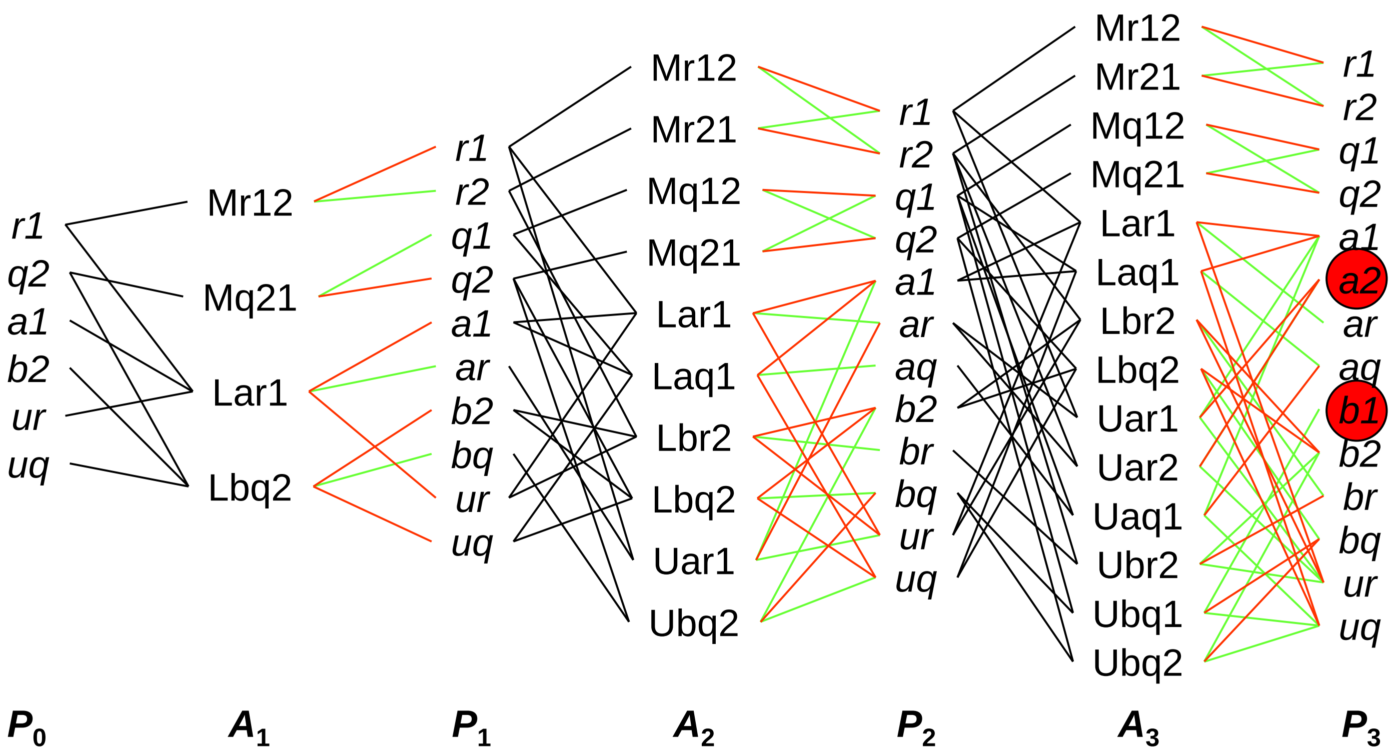
- room for improvement: use better heuristic function!

Planning Graph Heuristics

- basic idea: use reachability graph analysis as a heuristic for forward search
  - $P = (A, s_i, g)$ be a propositional planning problem and $G = (N, E)$ the corresponding planning graph
  - $g = \{g_1, \ldots, g_n\}$
  - g_k, $k \in [1,n]$, is reachable from s_i if there is a proposition layer P_g such that $g_k \in P_g$
  - in proposition layer P_m: if g_k not in P_m then g_k not reachable in m steps

- define (admissible) $h_{\mathrm{PG}}(g_k) = m$ for reachable $\{g_k\}$

Graphplan: Heuristic



goal: {*a2*, *b1*}

$h_{PG}(a2) = 3$

$h_{PG}(b1) = 3$

Overview

- Simple Planning Graph Heuristics
- **The FF (Fast Forward) Planner**

The FF Planner

- performs forward state-space search (A*)
- relaxed problem heuristic (h^{FF})
    - construct relaxed problem: ignore delete lists
    - solve relaxed problem (in polynomial time)
        - chain forward to build a relaxed planning graph
        - chain backward to extract a relaxed plan from the graph
    - use length of relaxed plan as heuristic value

Relaxed Planning Problem: Example

- move(*r*,*l*,*l'*)
  - precond: at(*r*,*l*), adjacent(*l*,*l'*)
  - effects: at(*r*,*l'*), ¬at(r,*l*)
- load(*c*,*r*,*l*)
  - precond: at(*r*,*l*), in(*c*,*l*), unloaded(*r*)
  - effects: loaded(*r*,*c*), ¬in(*c*,*l*), ¬unloaded(*r*)
- unload(*c*,*r*,*l*)
  - precond: at(*r*,*l*), loaded(*r*,*c*)
  - effects: unloaded(*r*), in(*c*,*l*), ¬loaded(*r*,*c*)

Computing h^{FF}: Relaxed Planning Graph

function computeRPG(A,s_i,g)

$F_0 \leftarrow s_i$; $t \leftarrow 0$

while $g \nsubseteq F_t$ **do**

$t \leftarrow t+1$

$A_t \leftarrow \{a \in A$ / precond(a) $\subseteq F_{t-1}\}$

$F_t \leftarrow F_{t-1}$

for all $a \in A_t$ **do**

$F_t \leftarrow F_t \cup$ effects$^+$(a)

if $F_t = F_{t-1}$ **then return** failure

return [F_0,A_1, F_1,…,A_t,F_t]

Computing h^{FF}: Extracting a Relaxed Plan

function extractRPSize($[F_0, A_1, F_1, \ldots, A_k, F_k], g$)

  if $g \nsubseteq F_k$ **then return** failure

  $M \leftarrow \max\{\text{firstlevel}(g_i, [F_0, \ldots, F_k]) \mid g_i \in g\}$

  for $t \leftarrow 0$ **to** M **do**

    $G_t \leftarrow \{g_i \in g \mid \text{firstlevel}(g_i, [F_0, \ldots, F_k]) = t)$

  for $t \leftarrow M$ **to** 1 **do**

    for all $g_t \in G_t$ **do**

      select a : firstlevel($a, [A_1, \ldots, A_t]$) $= t$ **and** $g_t \in \text{effects}^+(a)$

      for all $p \in \text{precond}(a)$ **do**

        $G_{\text{firstlevel}(p, [F0, \ldots, Fk])} \leftarrow G_{\text{firstlevel}(p, [F0, \ldots, Fk])} \cup \{p\}$

  return number of selected actions

End