

# COURSE "AUTOMATED PLANNING: THEORY AND PRACTICE"

## CHAPTER 15: PATTERN DATABASES

Teacher: **Marco Roveri** - marco.roveri@unitn.it

M.S. Course: Artificial Intelligence Systems (LM)

A.A.: 2023-2024

Where: DISI, University of Trento

URL: <https://bit.ly/3z0kGk8>



Last updated: Monday 13<sup>th</sup> November, 2023

# TERMS OF USE AND COPYRIGHT

## USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2023-2024.

## SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

## COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

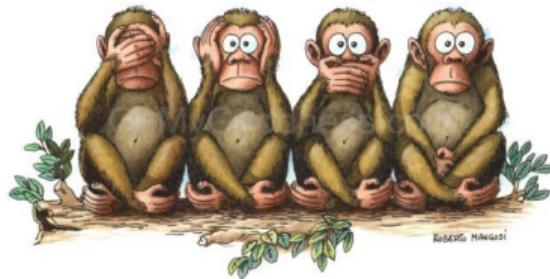
The material (text, figures) in these slides is authored by Jonas Kvarnström and Marco Roveri.

# PATTERN DATABASES: INTRODUCTION

- First a main idea behind pattern databases:
  - Let's care about few facts and ignore all others - everywhere
    - Goals, preconditions, effects, states

A form of relaxation...

With some special features!



OhMyGoodness.com

©<https://w-eu.ohmygoodness.com/reserved/cards/>

202004280553460.monkeys-dont-omg.jpg

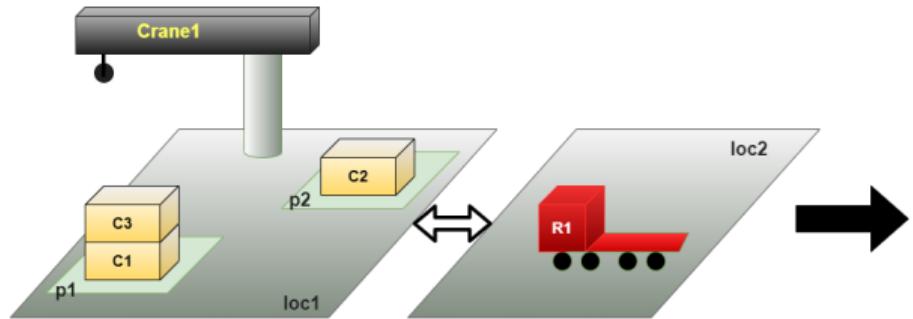
# PATTERN DATABASES: EXAMPLE

- Dock Worker Robots
  - Could choose to care about **container locations**
    - (in cont pile), (top container pile), (on container container), ...
  - **Ignore** robot locations, crane locations, location adjacency, ...

Ordinary state in problem P,  
All facts included!

States are "grouped together"

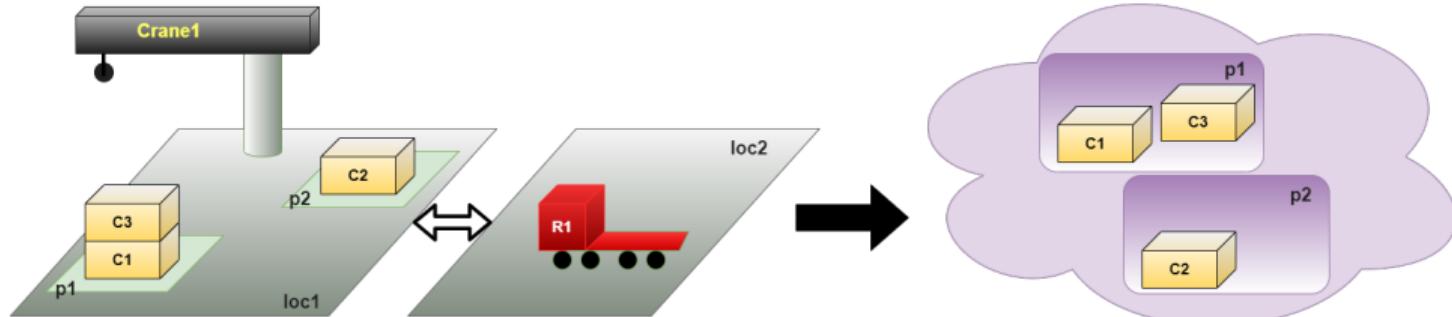
Abstract state in P', representing many states  
in P with different robot locations, ...



# PATTERN DATABASES: PLANNING IN PATTERNS

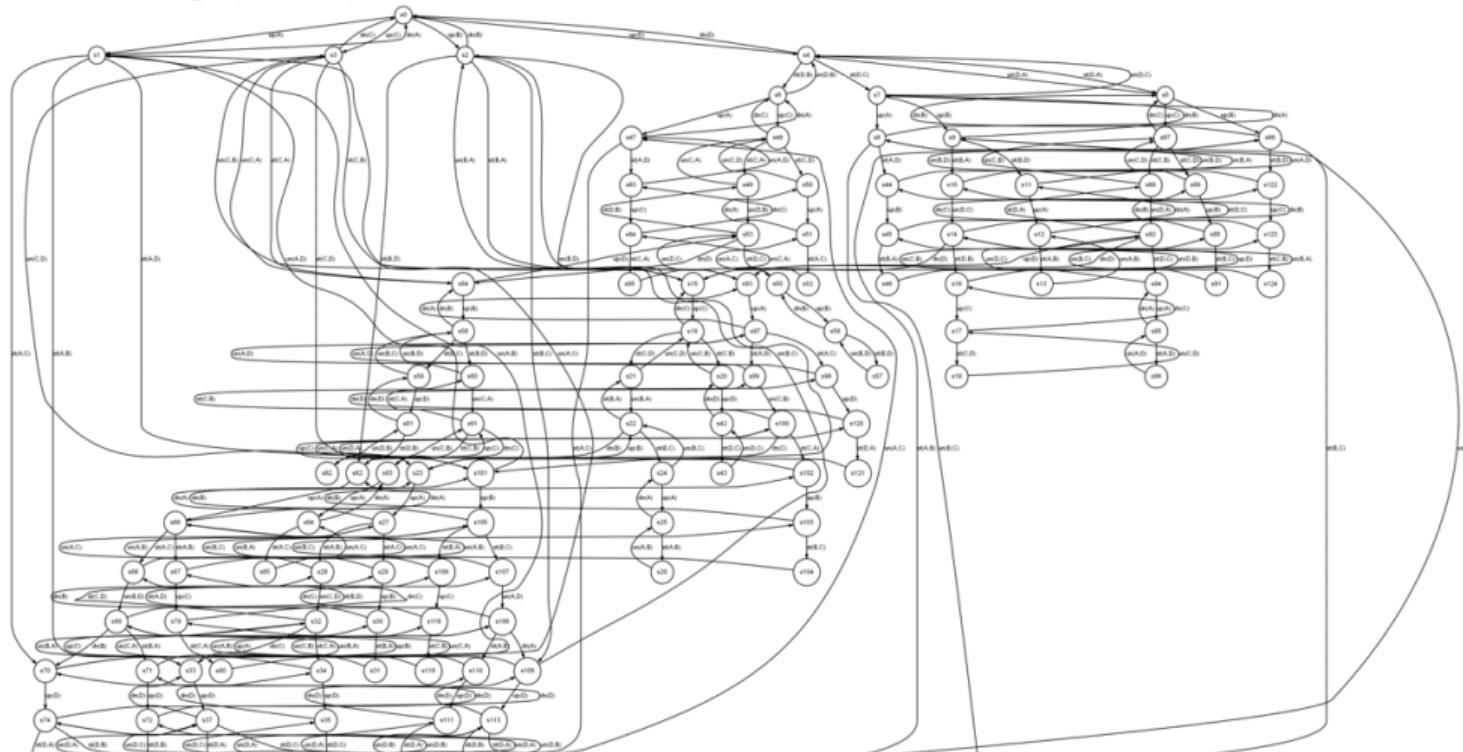
- In P' we (pretend that we) **can** use Crane1 at p1 to:
  - pick up c3 (should be possible)
  - place something on R1 (too far away, but we don't care)
  - place five containers on a single truck
- But we **can't**:
  - pick up c1 (we do care about pile ordering)
  - immediately place c1 below c2, ...

New paths  
to the goal!



# ACHIEVABLE STATES

- Consider the physically achievable states in the BW4:



# GROUND ATOMS

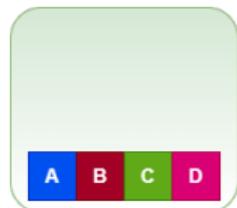
- All ground atoms (facts) in this problem instance:

(on A A)	(on A B)	(on A C)	(on A D)
(on B A)	(on B B)	(on B C)	(on B D)
(on C A)	(on C B)	(on C C)	(on C D)
(on D A)	(on D B)	(on D C)	(on D D)
(ontable A)	(ontable B)	(ontable C)	(ontable D)
(clear A)	(clear B)	(clear C)	(clear D)
(holding A)	(holding B)	(holding C)	(holding D)
(handempty)			

# A PATTERN

- Example: only consider 5 ground facts related to block A
  - "Pattern" :=  $p = \{ (\text{on } A \text{ } B), (\text{on } A \text{ } D), (\text{clear } A), (\text{ontable } A) \}$

- Initial state:



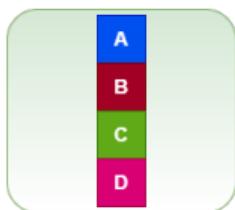
(ontable A)  
 (ontable B)  
 (ontable C)  
 (ontable D)  
 (clear A)  
 (clear B)  
 (clear C)  
 (clear D)  
 (handempty)



(ontable A)  
 (ontable B)  
 (ontable C)  
 (ontable D)  
 (clear A)  
 (clear B)  
 (clear C)  
 (clear D)  
 (handempty)

An  
 "abstract  
 state"

- Goal:



(clear A)  
 (on A B)  
 (on B C)  
 (on C D)  
 (ontable D)  
 (handempty)



(clear A)  
 (on A B)  
 (on B C)  
 (on C D)  
 (ontable D)  
 (handempty)

An  
 "abstract  
 goal"

# TRANSFORMED ACTIONS

- "Pattern" :=  $p = \{ (\text{on } A B), (\text{on } A D), (\text{clear } A), (\text{ontable } A) \}$
- Example action: (unstack A B)

- Before transformation

```
:precondition (and (clear A) (on A B))
:effect       (and (not (clear A)) (not (on A B))
                  (not (handempty)) (holding A) (clear B))
```

- After transformation

```
:precondition (and (clear A) (on A B))
:effect       (and (not (clear A)) (not (on A B)))
```

- Example action: (unstack C D)

- Before transformation

```
:precondition (and (clear C) (on C D))
:effect       (and (not (clear C)) (not (on C D))
                  (not (handempty)) (holding C) (clear D))
```

- After transformation

```
:precondition (and )
:effect       (and )
```

Loses **some** preconditions and effects!

Let's call this action resulting from pattern application action  $a \cap p$   
 not a set, but " $a$  restricted to  $p$ "!

Loses **all** preconditions and effects!  
 $\Rightarrow$  never used!

# PATTERNS, ABSTRACT STATES

- Given a pattern (set of ground facts)  $p$ 
  - A state  $s$  is represented by the abstract state  $s \cap p$

(clear A)  
 (on A B)  
 (on B C)  
 (on C D)  
 (ontable D)  
 (handempty)

 $\approx$ 

(clear A)  
 (on A B)  
 (ontable B)  
 (clear C)  
 (on C D)  
 (ontable D)  
 (handempty)

 $\approx$ 

(clear A)  
 (on A B)  
 (on B D)  
 (on D C)  
 (ontable C)  
 (handempty)

represented  
by a single  
abstract  
state

(clear A)  
 (on A B)

(clear A)  
 (ontable A)  
 (clear B)  
 (on B C)  
 (on C D)  
 (ontable D)  
 (handempty)

 $\approx$ 

(clear A)  
 (ontable A)  
 (holding B)  
 (clear C)  
 (on C D)  
 (ontable D)

 $\approx$ 

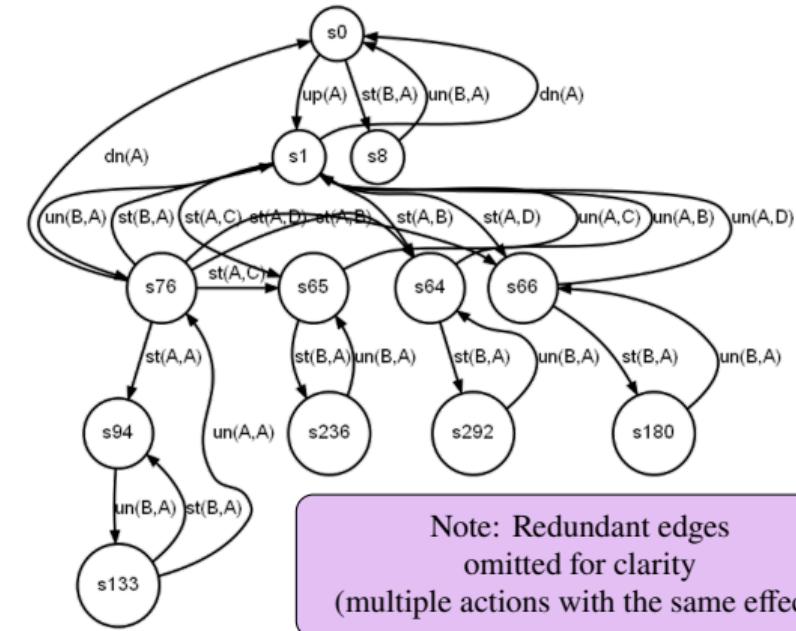
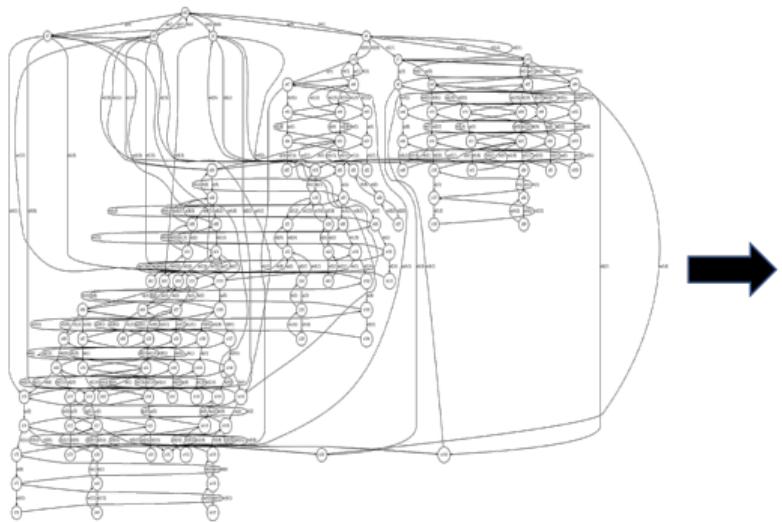
(clear A)  
 (ontable A)  
 (clear B)  
 (on B D)  
 (on D C)  
 (ontable C)  
 (handempty)

represented  
by a single  
abstract  
state

(clear A)  
 (ontable A)

# SMALLER STATE TRANSITION GRAPH

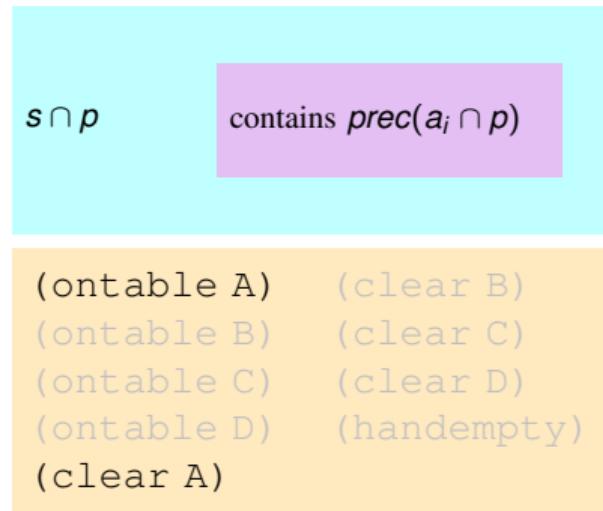
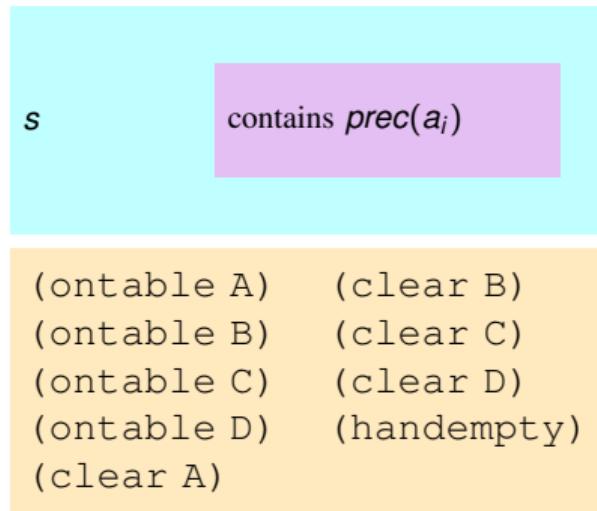
- Reachable state transition graph for the given pattern:
  - Current state:** Everything on the table, hand empty, all blocks clear
  - Abstract state:  $s_0 = \{ (\text{ontable } A), (\text{clear } A) \}$



# RELAXATION

- Why is this a relaxation?

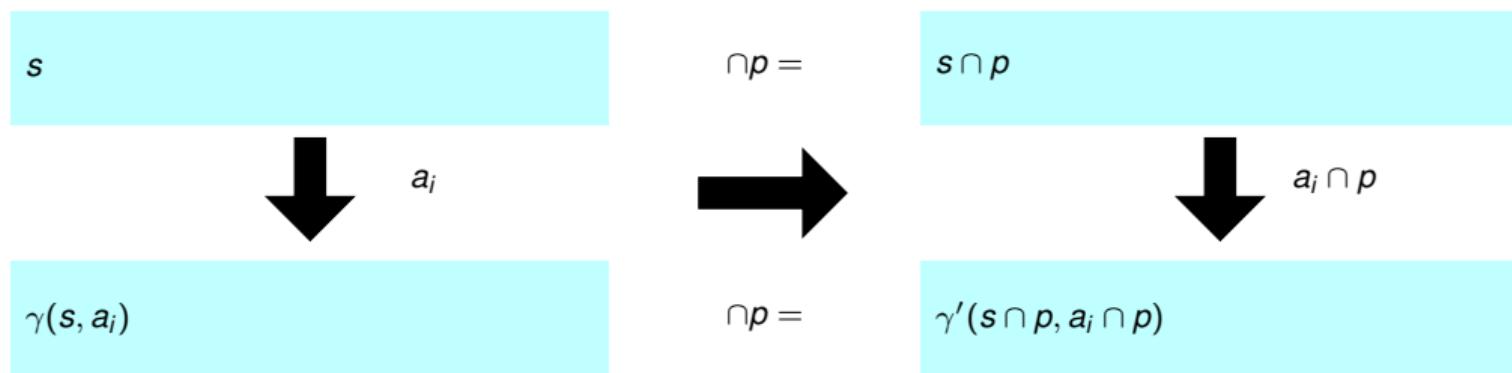
- Step 1: In  $s$  we can execute  $a_i$   $\implies$  in  $s \cap p$  we can execute  $a_i \cap p$



## RELAXATION (CONT.)

- If  $\gamma'$  is the state transition function for transformed action/states, then:

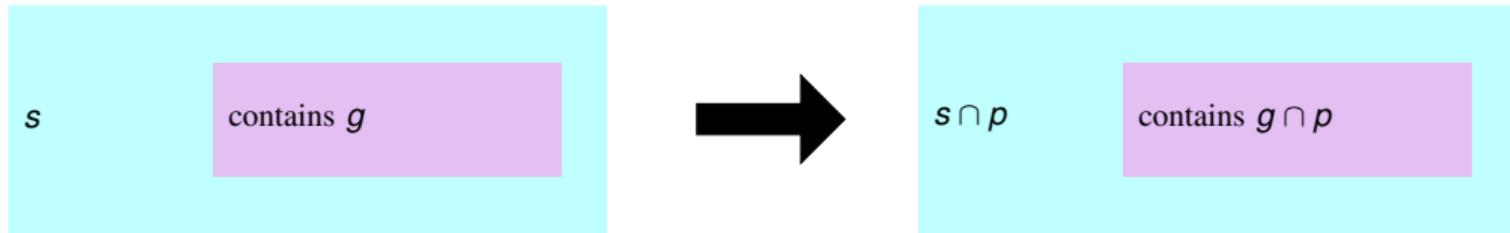
$$\gamma'(s \cap p, a_i \cap p) = \gamma(s, a_i) \cap p$$



⇒ Executable action sequences are preserved!

## RELAXATION (CONT.)

- If  $g \subseteq s$ , then  $g \cap p \subseteq s \cap p$



⇒ Solutions are preserved (but new solutions may arise!)

# PDB HEURISTIC: STATE VARIABLES

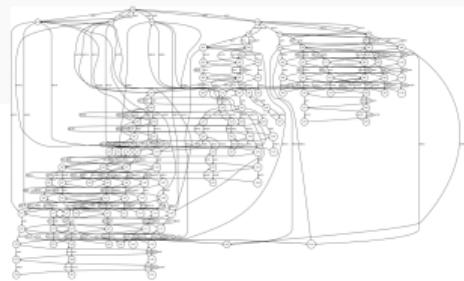
- For PDB heuristics, a **state variable representation** is useful
  - Typically:
    - Reduces the number of facts
    - Provides more information about which states are actually reachable!
  - **Model** problems using the state variable representation, or let planners **convert** automatically from predicate representation

# PDB HEURISTIC: STATE VARIABLES (CONT.)

- Repetition: Blocks world with 4 blocks
  - 536870912 states (reachable and unreachable) in the standard predicate representation
- But in all 125 states reachable from "all-on-table" (all "normal" states):
  - Block A satisfies exactly one of the following:
 

(holding A)	- held in the gripper
(clear A)	- at the top of the tower
(on B A)	- below B
(on C A)	- below C
(on D A)	- below D
  - $\Rightarrow$  remove those facts, introduce state variables

$\text{aboveA} \in \{\text{gripper}, \text{nothing}, \text{B}, \text{C}, \text{D}\}$



# PDB HEURISTIC: STATE VARIABLES (CONT.)

- Example, continued

- 536870912 states (reachable and unreachable) in the standard predicate representation
- 20000 states (reachable and unreachable) in state variable representation

```

aboveA ∈ {nothing, B, C, D, gripper}
aboveB ∈ {nothing, A, C, D, gripper}
aboveC ∈ {nothing, A, B, D, gripper}
aboveD ∈ {nothing, A, B, C, gripper}
posA   ∈ {on-table, other}
posB   ∈ {on-table, other}
posC   ∈ {on-table, other}
posD   ∈ {on-table, other}
hand   ∈ {empty, full}

```

The state variable translation is not part of the PDB heuristic!

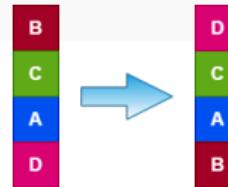
Using state variables is useful because PDBs work better with fewer "irrelevant states" in the state space

Allows structure to remain in the abstract search space:  
Preserves the fact that A can't be under B **and** under C)

Also useful when choosing facts: Ignore **where A is**, care about **where B is**

# PDB HEURISTIC: REWRITING THE PROBLEM

- Rewriting works as before
  - Suppose the pattern is { aboveB, aboveD, posB, posD }
- Rewrite the goal
  - Original: { aboveB=A, aboveA=C, aboveD=nothing, hand=empty }
  - Abstract: { aboveB=A, aboveD=nothing }
- Rewrite actions, removing some preconds / effects
  - (unstack A D) no longer requires aboveA=nothing
  - (unstack B C) still requires aboveA=nothing
- ...



```

aboveA ∈ {nothing, B, C, D, gripper}
aboveB ∈ {nothing, A, C, D, gripper}
aboveC ∈ {nothing, A, B, D, gripper}
aboveD ∈ {nothing, A, B, C, gripper}
posA ∈ {on-table, other}
posB ∈ {on-table, other}
posC ∈ {on-table, other}
posD ∈ {on-table, other}
hand ∈ {empty, full}

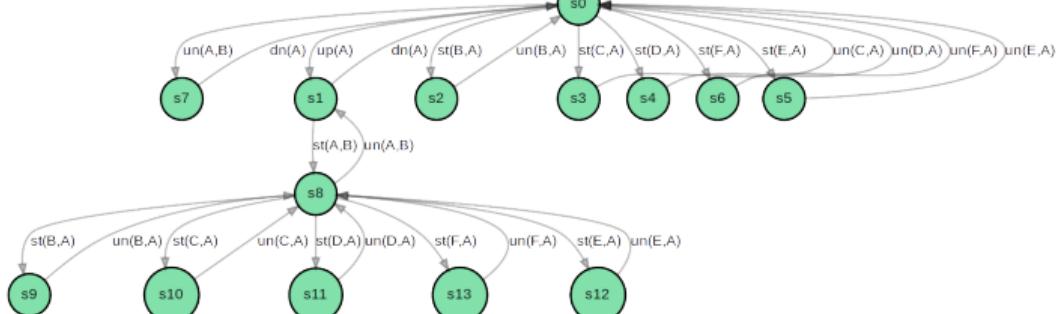
```

# PDB HEURISTIC: STATE SPACE SIZE

- Abstract states reachable from "all on table", by pattern  $p$  ...

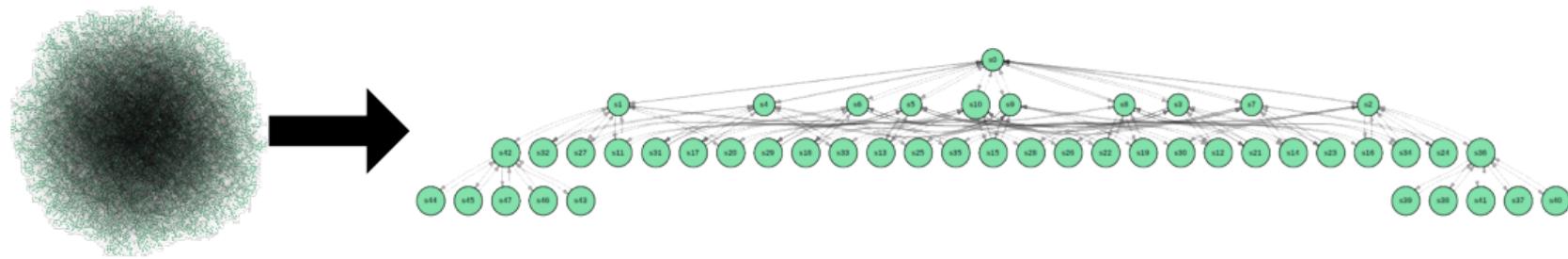
Blocks	All vars	$p=\{\text{aboveA}\}$	$p=\{\text{aboveA}, \text{posA}\}$	$p=\{\text{aboveA}, \text{aboveB}\}$
4	125	5	10	24
5	866	6	12	35
6	7057	7	14	48
7	65990	8	16	63
8	695417	9	18	80
9	8145730	10	20	99

Immediately  
(unstack A B)  
Don't care if  
aboveB=A



# PDB HEURISTIC: STATE SPACE SIZE (CONT.)

- For 6 blocks, originally 7057 reachable states
  - Pattern {aboveA, aboveB}  $\Rightarrow$  48 reachable states



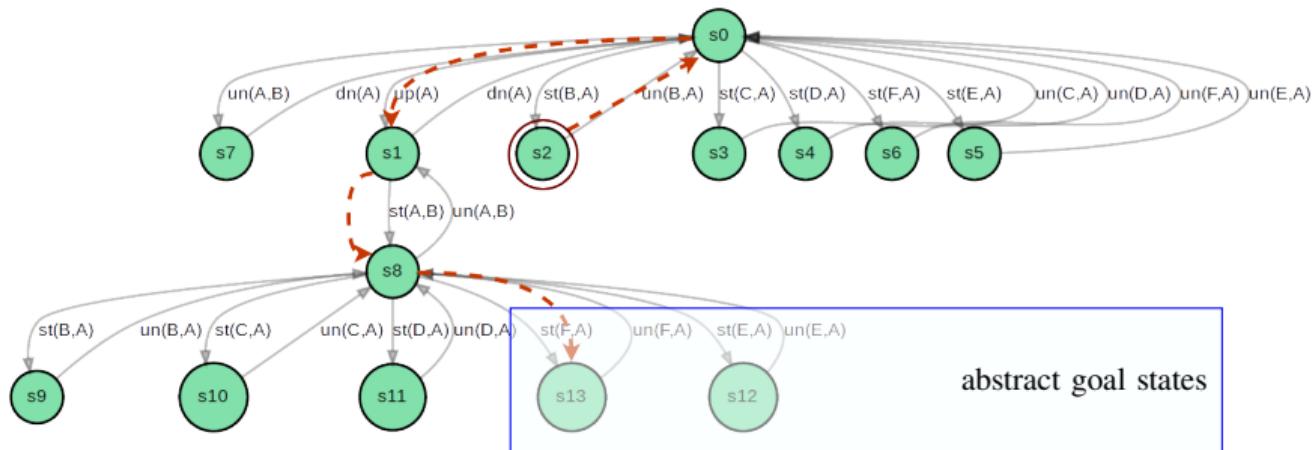
# PDB COMPUTATION: MAIN IDEA

- To calculate  $h_{pdb}(s)$  for a new state  $s$ :
  - Example: BW6, pattern {aboveA, posA}
- Convert  $s$  to *abstract* state
- Find **optimal** path to abstract goal state – in a much smaller search space
  - Current abstract state is  $s_2$

aboveA = B,  
posA = on-table,  
aboveB = nothing,...

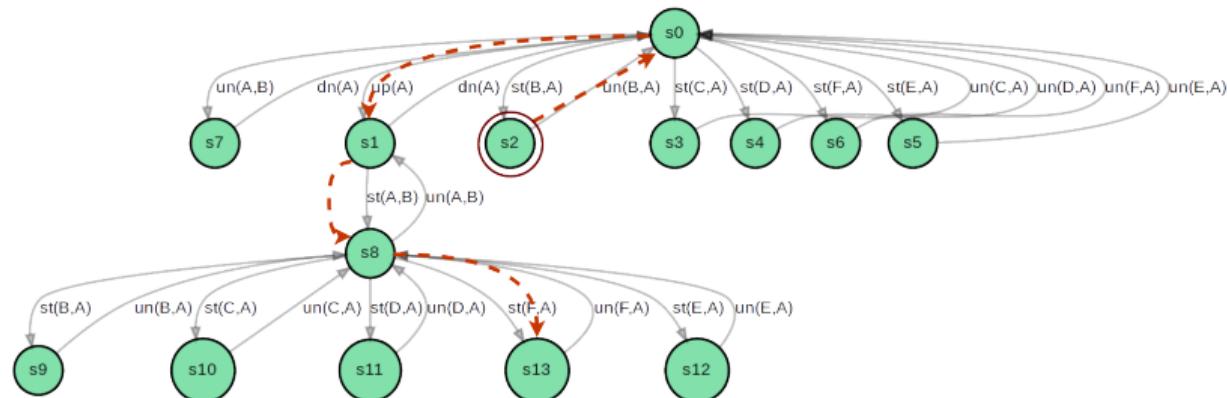
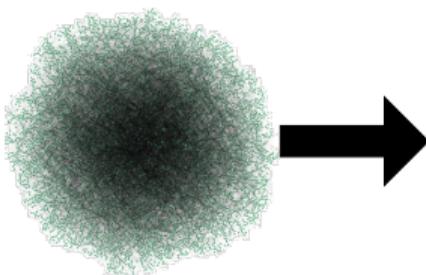


aboveA = B,  
posA = on-table



# TWEAK 1: CACHING

- Because we keep few state variables:
  - Many* real states map to the same abstract state
  - ⇒ An abstract state may be encountered **many** times during search
  - ⇒ **Cache** calculated costs



## TWEAK 2: DATABASES!

- Dijkstra efficiently finds optimal paths from **all** abstract states
  - $\Rightarrow$  Precalculate **all** heuristic values for each pattern
  - Store in a look-up table – a **database**

Second main idea!

# PREPROCESSING

- Preprocessing step 1 (BW 6, pattern { aboveA, posA})
  - Create the initial abstract state...

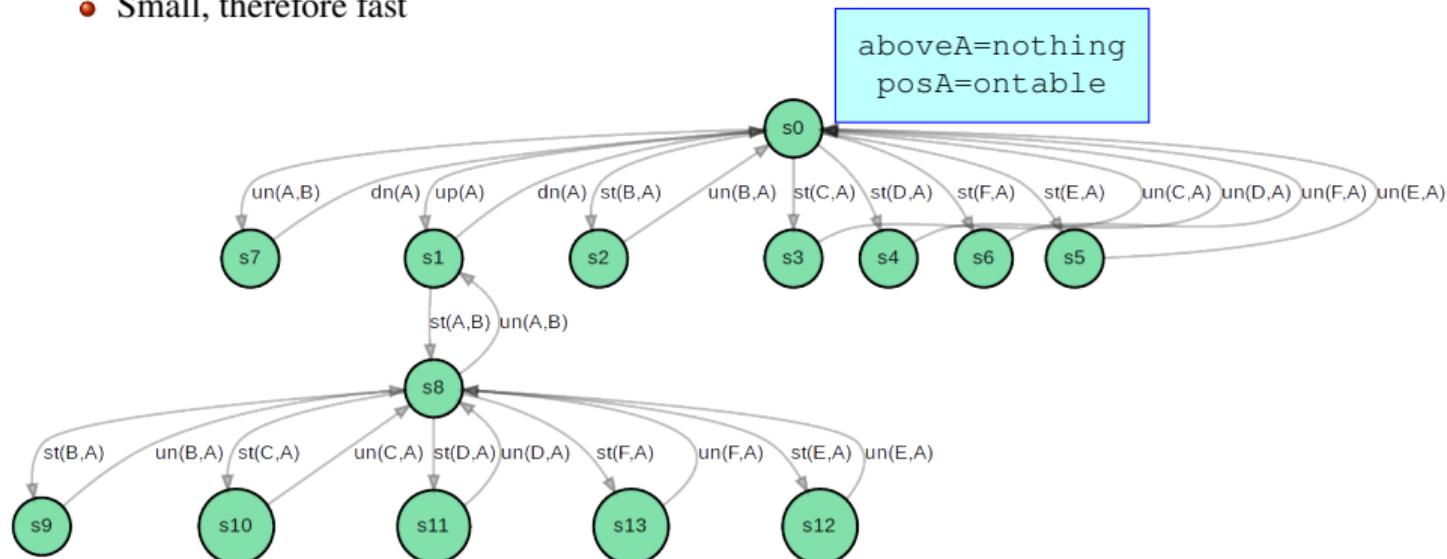
```
aboveA=nothing  
aboveB=nothing  
aboveC=nothing  
aboveD=nothing  
aboveE=nothing  
aboveF=nothing  
posA=ontable  
posB=ontable  
posC=ontable  
posD=ontable  
posE=ontable  
posF=ontable  
hand=empty
```



```
aboveA=nothing  
aboveB=nothing  
aboveC=nothing  
aboveD=nothing  
aboveE=nothing  
aboveF=nothing  
posA=ontable  
posB=ontable  
posC=ontable  
posD=ontable  
posE=ontable  
posF=ontable  
hand=empty
```

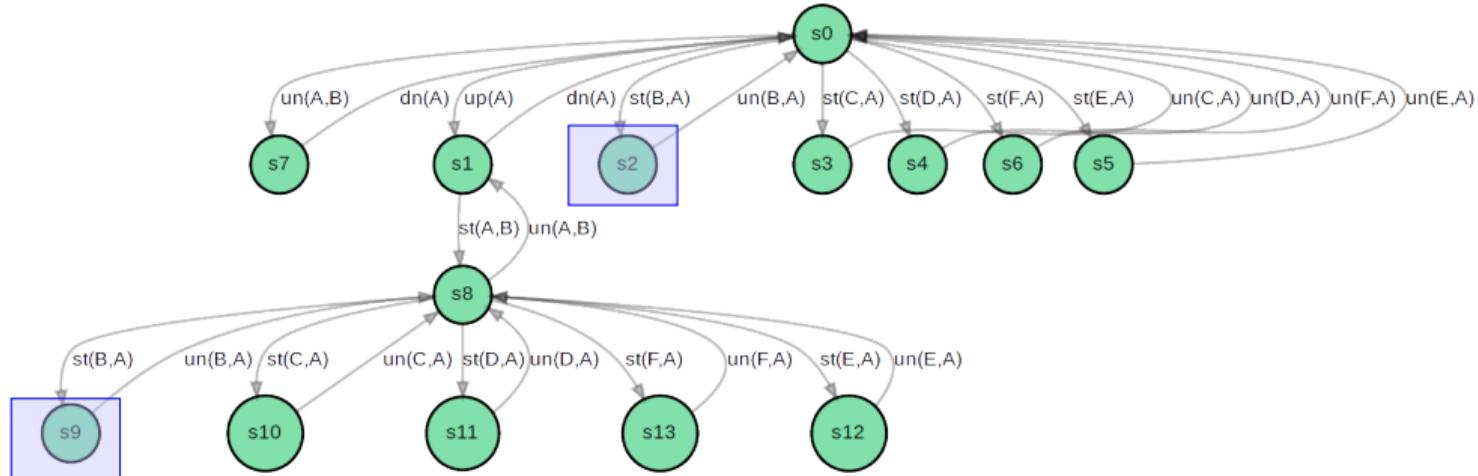
# PREPROCESSING (CONT.)

- Preprocessing step 2
  - Find all **reachable** abstract states
    - Start in the abstract initial state ( $s_0$  below)
    - Apply all applicable actions to all states you find (DFS, BFS, ...)
    - Small, therefore fast



## PREPROCESSING (CONT.)

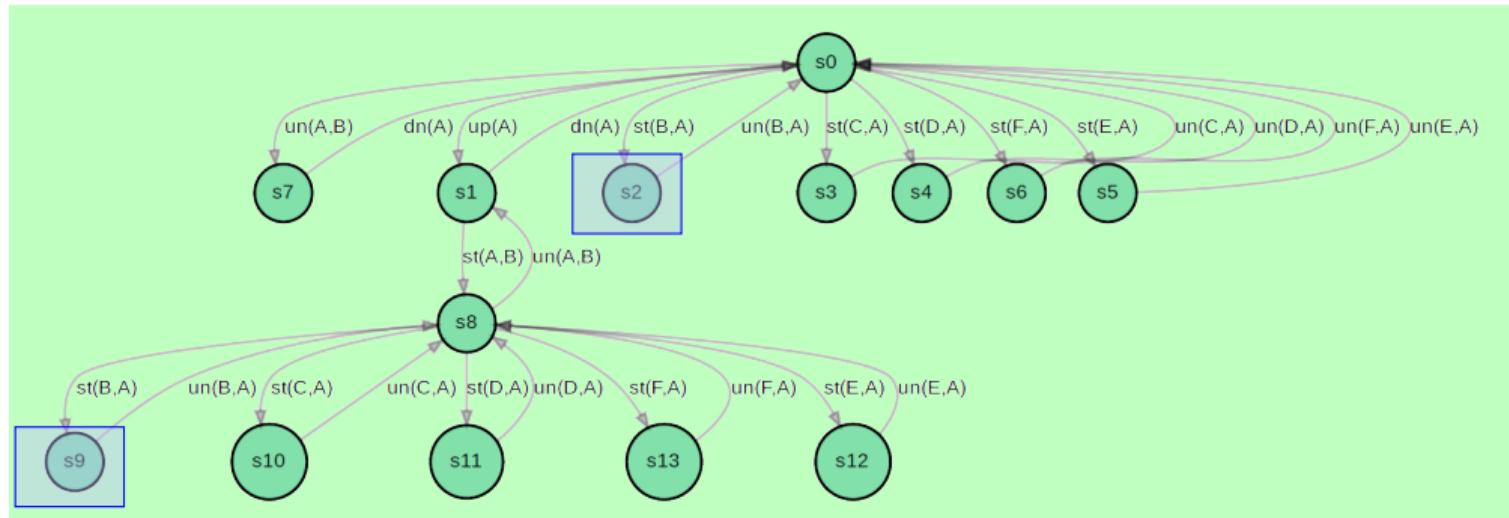
- Preprocessing step 3: Which abstract states satisfy the **abstract goal**?
  - Real goal: { aboveA=B, aboveB=C, aboveC=D, aboveD=E, aboveE=F }
  - Abstract goal: { aboveA=B }
  - Satisfied in **two** of the reachable states



## PREPROCESSING (CONT.)

- Preprocessing step 4: Compute the database

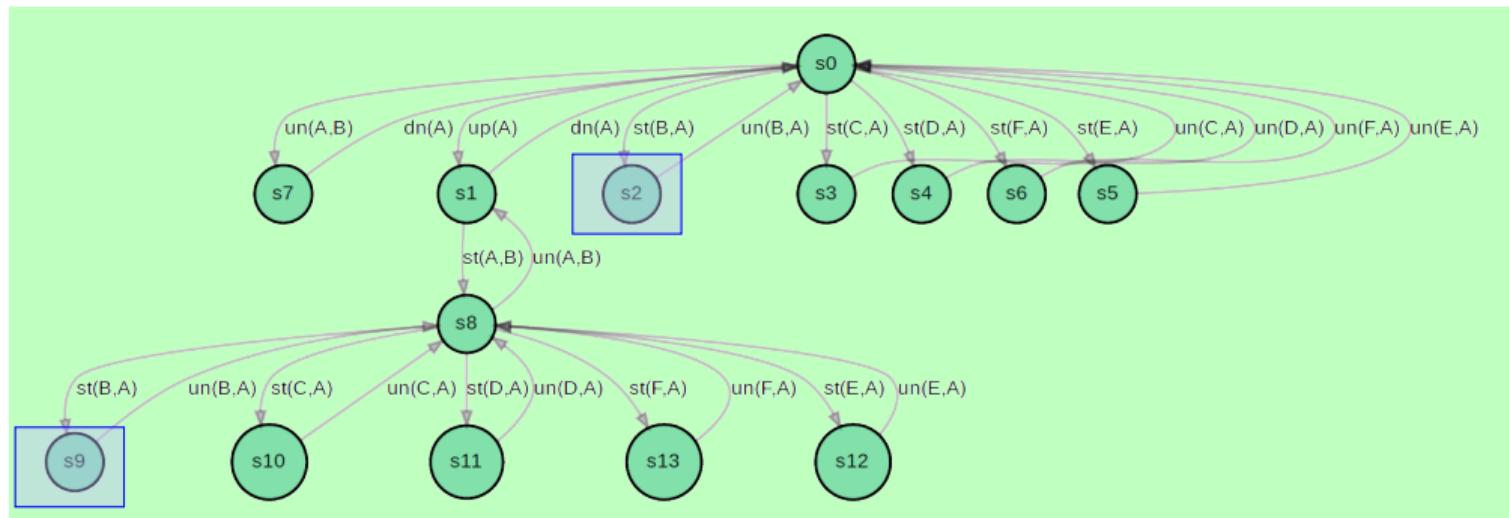
- For every abstract state reachable from the abstract initial state,
- find a cheapest path to any abstract goal state



# PREPROCESSING (CONT.)

- More efficient computation:

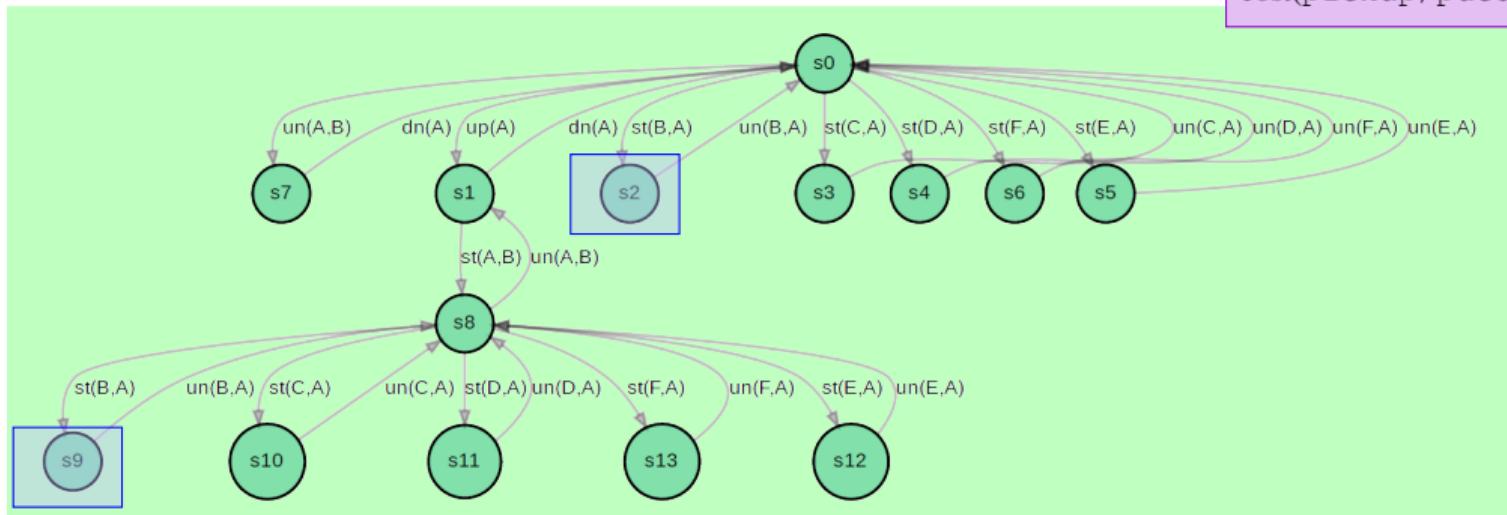
- Start with cost=0 for all abstract goal states
- Search backward from these states – single call to Dijkstra
  - Queue initialized with multiple nodes/states of cost 0 ( $s_2, s_9$ )
  - Edges followed backward (in explicitly computed graph, no  $\gamma^{-1}$  necessary)



## PREPROCESSING (CONT.)

- Example: Priority queue contains  $< s_2 / \text{cost} = 0, s_9 / \text{cost} = 0 >$ 
  - Pick  $s_2$ 
    - A "successor" is  $s_0$ , reached backwards by (stack B A) of cost 2;
    - $s_0$  inserted in the queue with cost 2
  - Priority queue contains  $< s_9 / \text{cost} = 0, s_0 / \text{cost} = 2 >$ , pick  $s_9$ 
    - A successor is  $s_8$  ...

Assuming  
 $\text{cost(stack/unstack)}=2$ ,  
 $\text{cost(pickup/putdown)}=1$



# PDB HEURISTICS: DATABASES

- Database:
    - Stores the cost found by Dijkstra for every **abstract state**  $s$ 
      - Cost is **optimal** within the relaxed problem
      - Cost is **admissible** for the "real" problem

# PDB HEURISTICS IN FORWARD SEARCH

- Step 1: **Automatically** generate a pattern
  - A selection of state variables to consider
  - Choosing a **good** pattern is a **difficult problem!**
    - Different approaches exist ...
- Step 2: Calculate the pattern database
  - As already discussed

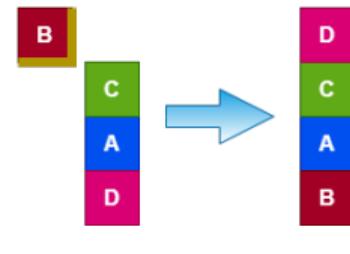
# PDB HEURISTICS IN FORWARD SEARCH (CONT.)

- Step 3: Forward search in the **original problem**

- For each new successor state  $s_1$ , calculate heuristic value  $h_{pdb}(s_1)$ 
  - Example:  $s_1 = \{ \text{aboveD=A, aboveA=C, aboveC=nothing, aboveB=gripper, posA=other, posB=other, posC=other, posD=ontable, hand=full} \}$

- Convert this to an abstract state

- Example:  $s'_1 = \{ \text{aboveD=A, aboveB=gripper, posB=other, posD=ontable} \}$
- Use the database to quickly look up  $h_{pdb}(s'_1) = \text{the cost of reaching the nearest abstract goal from } s'_1$



<pre> abstract state aboveB = gripper, aboveD = A, posB = other, posD = on-table aboveB = gripper, aboveD = A, posB = other, posD = other ... </pre>	$\Rightarrow$ cost $n_1$ $n_2$ $\Rightarrow$
--	---

# COMPUTATION TIME

- How close to  $h^*(n)$  can an admissible PDB-based heuristic be?
  - Wrong question!
    - A pattern can include all state variables  $\Rightarrow$  equal to  $h^*(n)$

Then computing the database takes too much time...

How much time?

Dijkstra's algorithm:  
 $O(|E| + |V| \log(|V|))$

$|V| =$  number of reachable  
 abstract states

# of reachable abstract:  
 Exponential in # of  
 selected state variables

Exponential in the number of state variables in the pattern...

Possible state variables: Linear in problem size (length of PDDL file, ...)  
 Select all  $\Rightarrow$  PDB computation exponential in problem size

## COMPUTATION TIME (CONT.)

We want: **Polynomial** in the problem size (length of PDDL file, ...)

Counteract **this**:

Exponential in the number of state variables in the pattern ...

Using **this**:

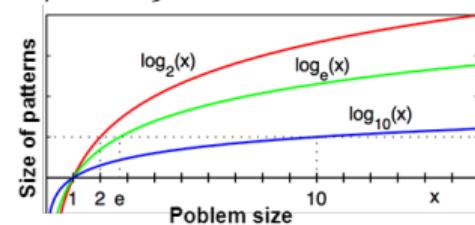
Select a **logarithmic** number of state variables – growing as  $\log(\text{problem size})$

Computation time:  $O(2^{\log(\text{problemsize})})$ , which is polynomial in problem size

# ASYMPTOTIC ACCURACY: SINGLE PDB HEURISTIC

- Assume (as before) an *infinite* set of problems  $\{P_i | i \geq 0\}$

- Assume a **strategy** for selecting a single pattern for each problem
  - So that pattern size grows at most as  $O(\log(k))$  for problem size  $k$



- What is the best **accuracy guarantee** any strategy can give?
  - $h(n) \leq$  cost of reaching the *most expensive subgoal* of size  $O(\log(k))$

Subgoal size is not constant but grows with problem size  $k$  – good!

But still,  $\log(k)$  grows much slower than  $k$  ...

- For any given single pattern, **asymptotic** accuracy is 0 in many domains
  - As before, *practical results* are usually better!

# PDB HEURISTICS: GRIPPER EXAMPLE

- A common restricted gripper domain:
  - One robot with two grippers
  - Two rooms
  - All  $n$  balls originally in the first room
  - Objective: All balls in the second room

Compact state variable representation:

$\text{loc(ball}_k\text{)} \in \{\text{room1, room2, gripper1, gripper2}\}$   
 $\text{loc(robot)} \in \{\text{room1, room2}\}$

$2 * 4^n$  states, some unreachable

Some possible patterns for  $k \geq 1$  balls:

$\{\text{loc(ball}_1\}\}$	$\implies$	4 abstract states
$\{\text{loc(ball}_1\}, \text{loc(robot)}\}$	$\implies$	8 abstract states
$\{\text{loc(ball}_i\} \mid i \leq k\}$	$\implies$	$4^k$ abstract states
$\{\text{loc(ball}_i\} \mid i \leq \log(k)\}$	$\implies$	$4^{\log(k)}$ abstract states



## BW4: SUBPROBLEM 2

- Subproblem 2: Some facts related to B

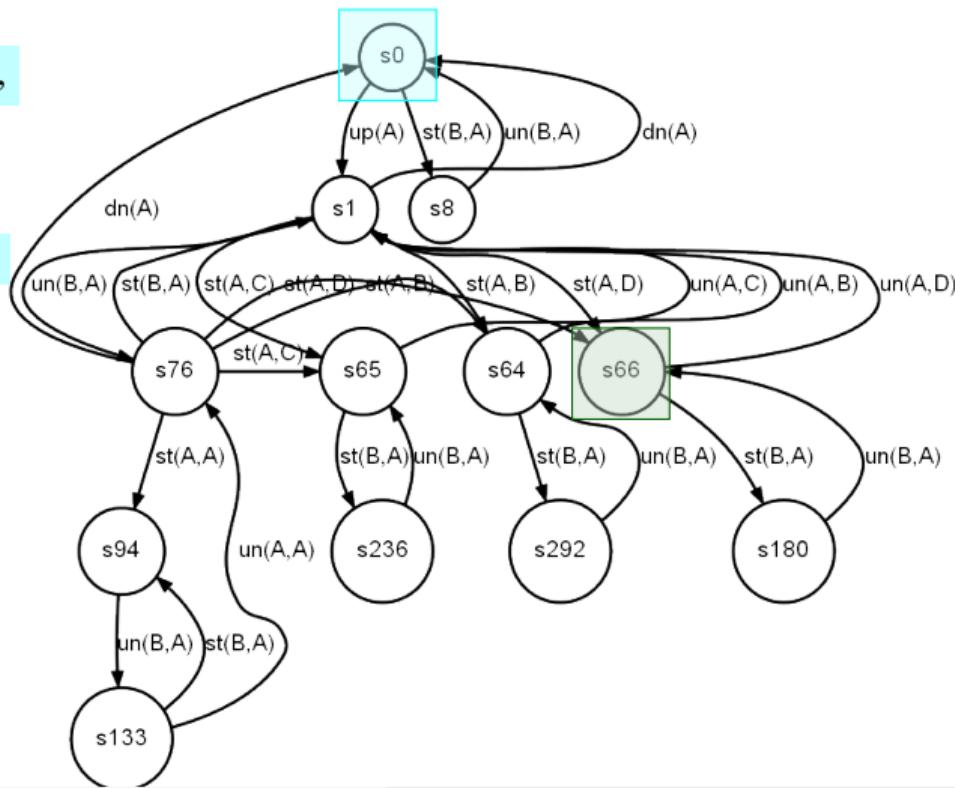
- Current state:** Everything on the table,  
hand empty, all blocks clear

- Abstract state:**  
 $\{(\text{ontable } B), (\text{clear } B)\}$

- Goal state:** A on B on C on D

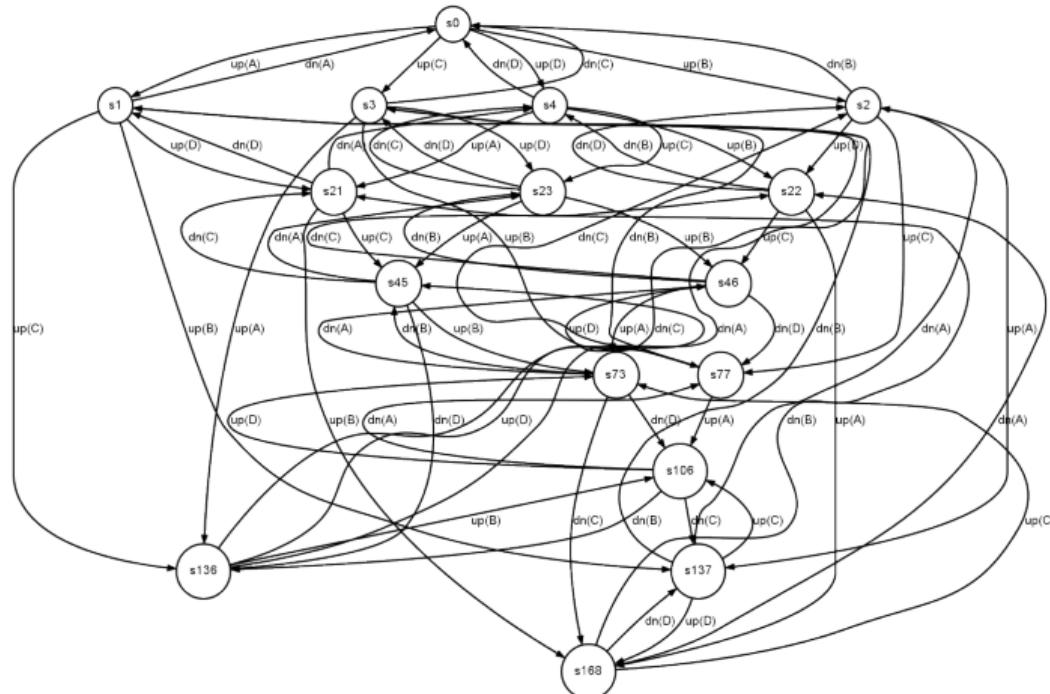
- Abstract goal:**  $\{(\text{on } B \text{ } C)\}$

- Find a path, compute its cost



## BW4: SUBPROBLEM 3

- Subproblem 3: Only consider `(holding ?x)` facts...
    - Not **handempty**, so can hold many blocks



# IMPROVING PDBs

- For each pattern, compute a **separate pattern database**
  - Each such cost is an admissible heuristic

What then? Do we sum the costs?

Possibly in satisficing planning – but would generally not be admissible...

Pattern 1, aboveB: I need  
(stack A B), cost 2

Pattern 2, aboveC: I need  
(stack B C), cost 2

Sum:  $2+2=4$

Could have used (buildTower A B C), cost 3  
Only detectable if we consider the **whole** problem

But the **maximum** of two admissible heuristics is also admissible!  
And polynomial time:  $k$  patterns require  $k$  computations

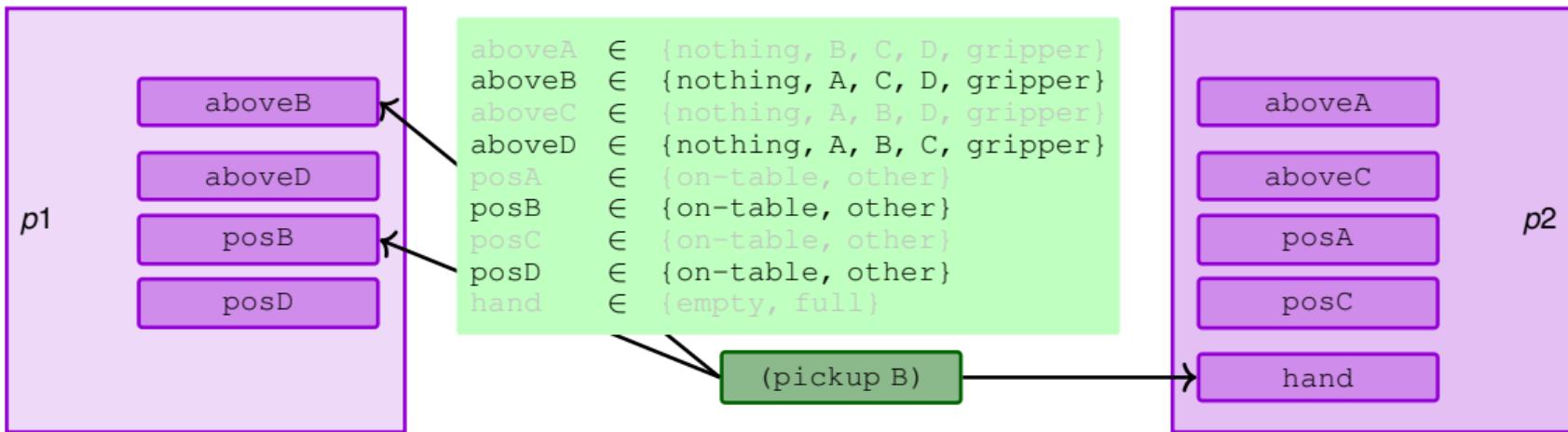
- What is the new level of **accuracy**?
  - Still 0... asymptotically | worst case
  - But this can still work well in practice!

# ADDITIVE PDB HEURISTICS

- To create **admissible** heuristics by **summing** PDB heuristics:
  - Each fact should **be** in **at most one** pattern
  - Each action should **affect** facts in **at most one** pattern
- $\Rightarrow$  **Additive** pattern database heuristics

## ADDITIVE PDB HEURISTICS (CONT.)

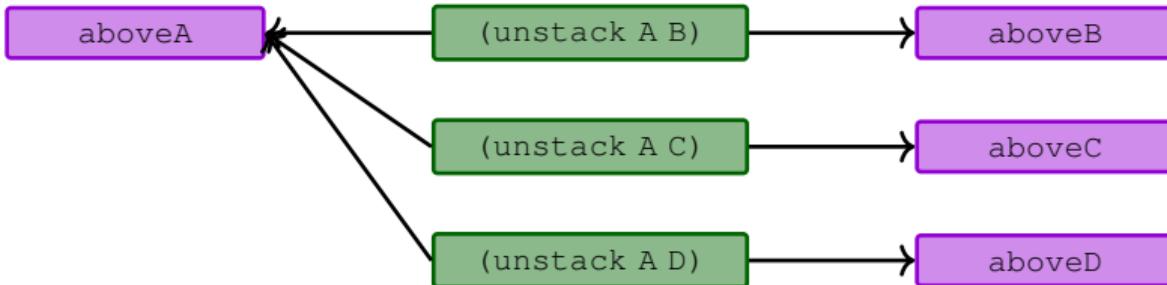
- BW: Is  $p1 = \{\text{facts in even rows}\}$ ,  $p2 = \{\text{facts in odd rows}\}$  *additive*?
  - No! (*pickup B*) affects  $\{\text{aboveB}, \text{posB}\}$  in  $p1$ ,  $\{\text{hand}\}$  in  $p2$



One potential problem:  
 Both patterns could use `(pickup B)` in their optimal solutions  
 $\Rightarrow$  sum counts this twice! This is what we're trying to avoid...

## ADDITIVE PDB HEURISTICS (CONT.)

- BW: Is  $p1 = \{\text{aboveA}\}$ ,  $p2 = \{\text{aboveB}\}$  additive?
  - No! ( $\text{unstack A B}$ ) affects  $\{\text{aboveB}\}$  in  $p1$ ,  $\{\text{aboveA}\}$  in  $p2$
  - True for all combination of  $\text{aboveX}$ !

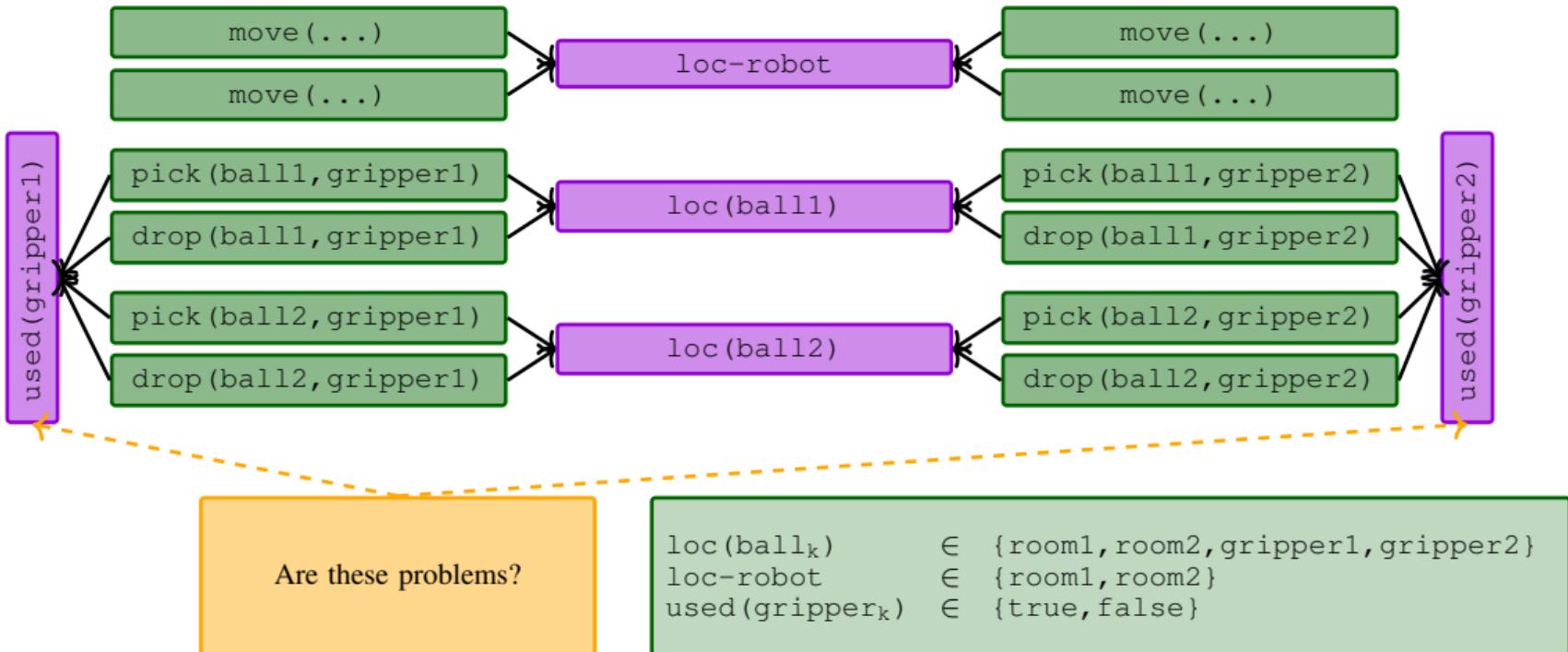


- An additive PDB heur. could use **one** of these:
  - $p1 = \{\text{aboveA}\}$
  - $p1 = \{\text{aboveA}, \text{aboveC}, \text{aboveD}\}$
  - ...
- Can't have two separate patterns  $p1, p2$  both of which include an  $\text{aboveX}$ 
  - Those  $\text{aboveX}$  will be directly connected by some  $\text{unstack}$  action

This formulation of the Blocks World is "connected in the wrong way" for this approach to work well

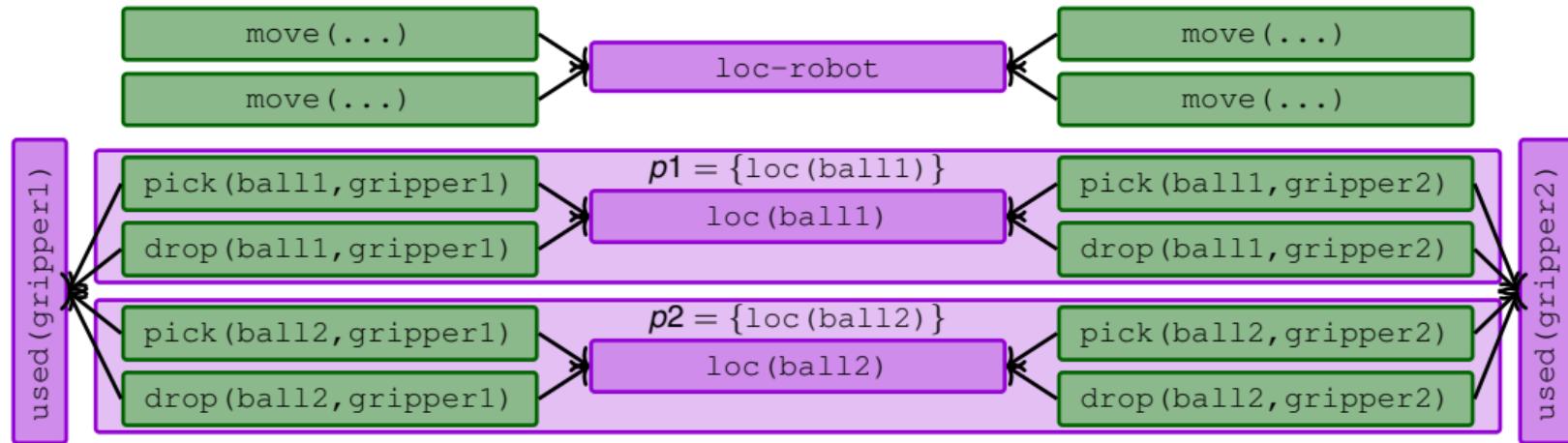
# ADDITIVE PDB HEURISTICS (CONT.)

- "Separating" patterns in the Gripper domain:



# ADDITIVE PDB HEURISTICS (CONT.)

- No problem: We don't have to use **all** variables in patterns!



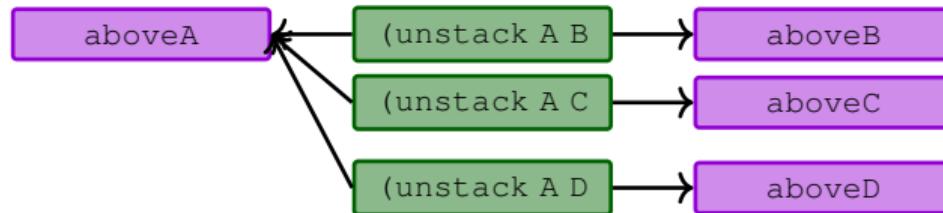
For each pattern we chose one **variable**

Then we have to include **all actions** affecting it

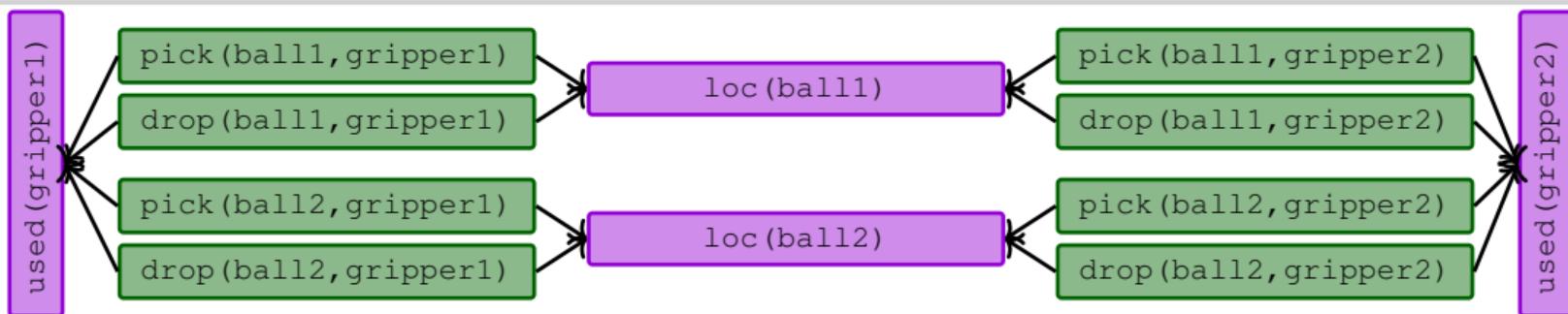
The other variables those actions affect [used(...)] don't have to be part of any pattern!

# ADDITIVE PDB HEURISTICS (CONT.)

- Notice the difference in structure!



BW: Every pair of  $\text{aboveX}$  facts has a *direct connection* through an action

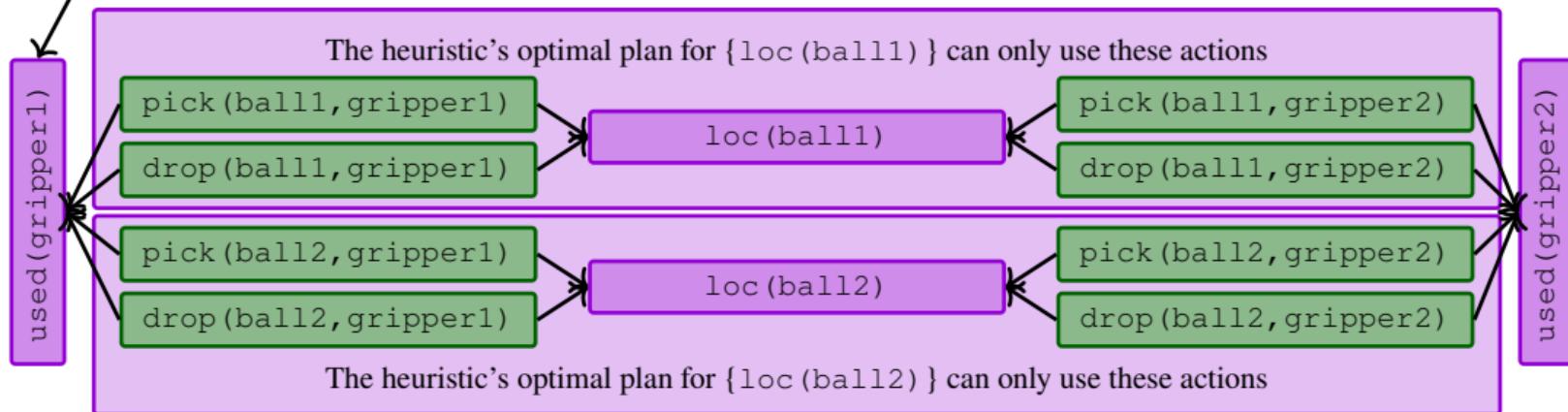


Gripper: No pair of  $\text{loc}()$  facts has a *direct connection* through an action

# ADDITIVE PDB HEURISTICS (CONT.)

- When every action affects facts in at most one pattern:
  - The subproblems we generated are completely disjoint
    - They achieve **different aspects** of the goal
    - Optimal solutions **must** use **different actions**

The heuristic never tries to generate optimal plans for `used(gripper1)` – we have not included it in any pattern



## ADDITIVE PDB HEURISTICS (CONT.)

- Avoids the overestimation problem

Problem:

Goal: p and q

A1: effect p

A2: effect q

A3: effect p and q // BuildTower

To achieve p    Heuristic uses A1

To achieve q    Heuristic uses A2

Sum of costs is 4 – optimal cost is 3, using A3

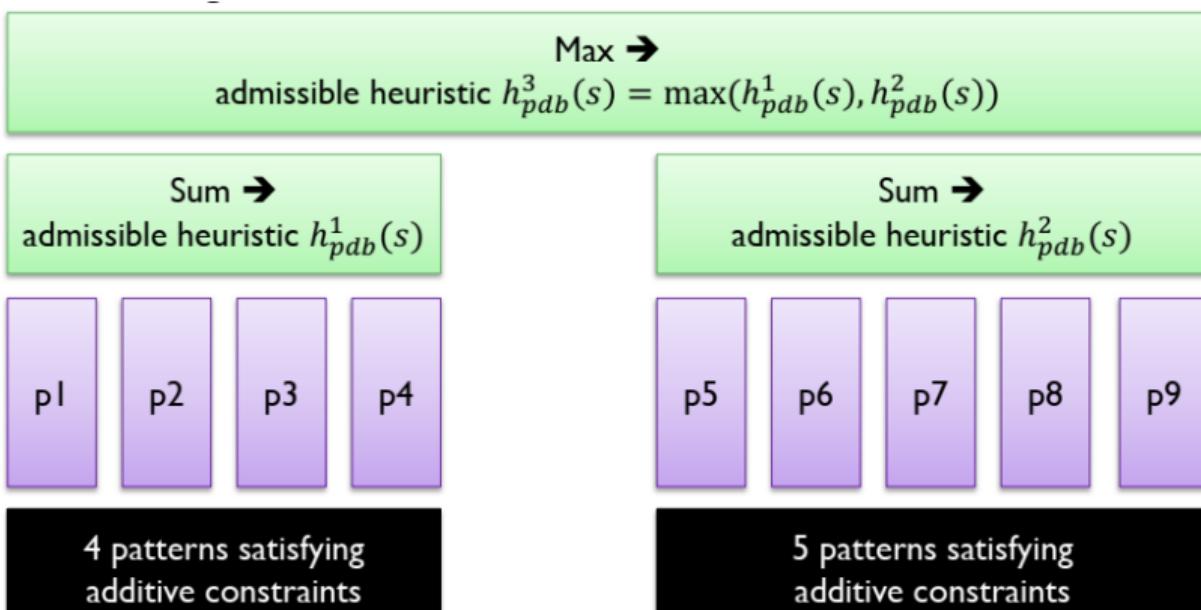
This cannot happen

when every action affects facts in  
at most one pattern

- $\implies$  The costs are additive for multiple patterns
- $\implies$  Adding costs from multiple heuristics yields an admissible heuristic!

## ADDITIVE PDB HEURISTICS (CONT.)

- Can be taken one step further...
  - Suppose we have several sets of additive patterns:
    - Can calculate an admissible heuristic from **each** additive set, then take the **maximum** of the results as a **stronger** admissible heuristic



## ADDITIVE PDB HEURISTICS (CONT.)

- How close to  $h^*(n)$  can an **additive** PDB-based heuristic be?
  - For additive PDB heuristics with a single sum, **asymptotic accuracy** as problem size approaches infinity...

## ADDITIVE PDB HEURISTICS (CONT.)

- In Gripper:

- In state  $s_n$  there are  $n$  balls in room1 and no balls are carried
- Additive PDB heuristic  $h_{add}^{PDB}(s_n)$ 
  - Use a separate pattern for each ball location variable  $\text{loc(ball}_k\text{)}$
  - For each pattern/ball, the optimal PDB cost is 2
    - $\text{pick(ball, room1, gripper1)} : \text{loc(ball)} = \text{room1} \implies \text{loc(ball)} = \text{gripper1}$
    - $\text{drop(ball, room2, gripper1)} : \text{loc(ball)} = \text{gripper1} \implies \text{loc(ball)} = \text{room2}$

$$\bullet h_{add}^{PDB}(s_n) = \text{sum for } n \text{ balls} = 2n$$

- Real cost:

- Using both gripper
  - $\text{pick, pick, move(room1, room2), drop, drop, move(room2, room1)}$
- Repeat  $n/2$  times, total cost  $\approx 6n/2 = 3n$
- $\implies$  Asymptotic accuracy  $2n/3n = 2/3$

## ADDITIVE PDB HEURISTICS (CONT.)

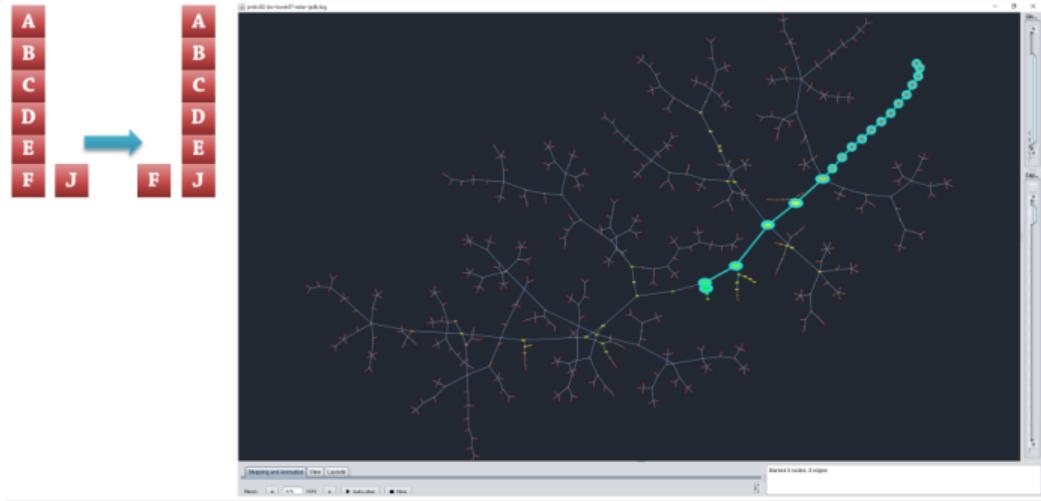
- What quality **guarantees** can an **additive** PDB heuristic give?
  - For additive PDB heuristics with a single sum, **asymptotic accuracy** (fraction of  $h^*$ ) as problem size approaches infinity:

	$h^+$ (too slow)	Additive PDB
Gripper	2/3	2/3
Logistics	3/4	1/2
Blocks world	1/4	0
Miconic-STRIPS	6/7	1/2
Miconic-Simple-ADL	3/4	0
Schedule	1/4	1/2
Satellite	1/2	1/6

- Only guaranteed if the planner finds the best combination of patterns!
  - This is a very difficult problem in itself!
- But as usual, this is a worst-case analysis...

# EXAMPLE

bw-tower07-astar-ipdb: Only 7 blocks, A\* search, based on PDB variation



- Blind A\*: 43150 states calculated, 33436 visited
- A\* + goal count: 6463 states calculated, 3222 visited
- A\* + iPDB: 1321 states calculated, 375 visited

No heuristic is perfect – visiting some additional states is fine!

# REFERENCES I

- [1] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. ISBN 9781608459698. doi: 10.2200/S00513ED1V01Y201306AIM022. URL <https://doi.org/10.2200/S00513ED1V01Y201306AIM022>.
- [2] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004. ISBN 978-1-55860-856-6.
- [3] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN 978-1-107-03727-4. URL <http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB>.