
Linköping University

# Automated Planning

## Pattern Databases:
## "Who cares about that?"

Jonas Kvarnström
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2019

# PDB 1: Introduction



jonkv@ida

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

**A form of relaxation…**

With some special features (later!)

# PDB 2: Dock Worker Robots



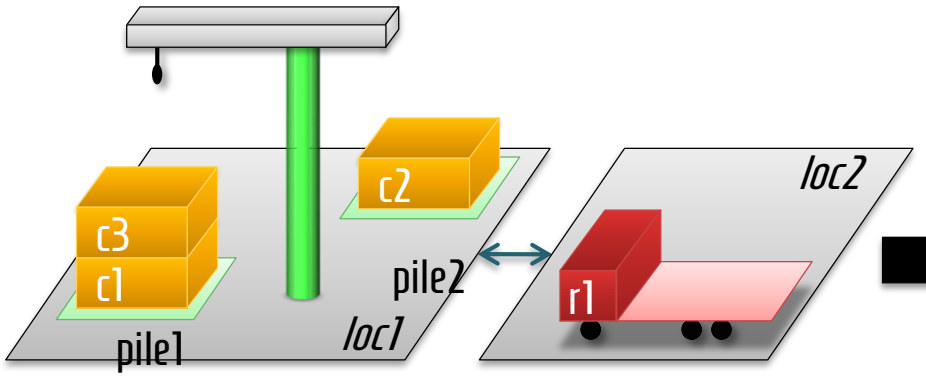
jonkv@ida

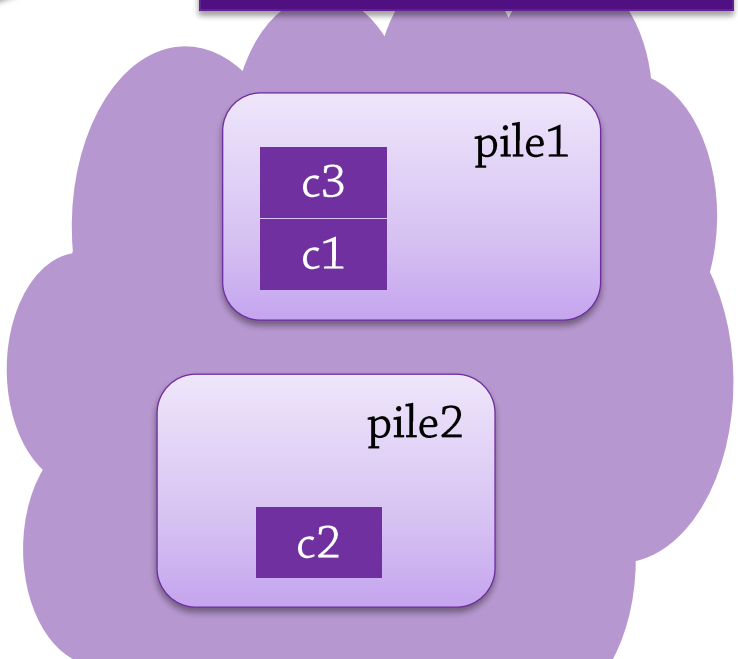
- Example from Dock Worker Robots
  - Could choose to care about **container locations**
    - *in(container, pile), top(container,pile), on(c1,c2), …*
  - **Ignore** robot locations, crane locations, location adjacency, …

**States are "grouped together"!**

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

Ordinary state in problem P, all facts included



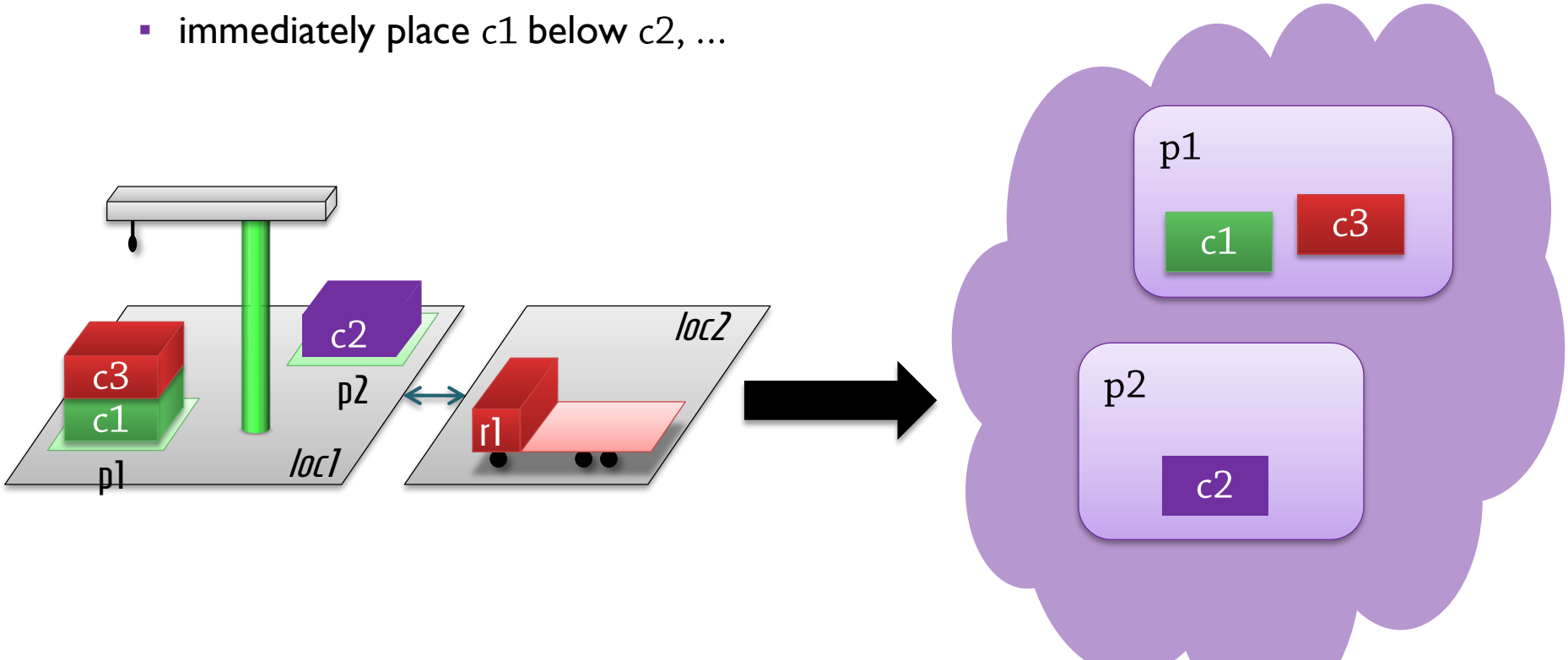
# PDB 3: Planning in Patterns



jonkv@ida

- In P' we (pretend that we) **can** use the crane 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!**



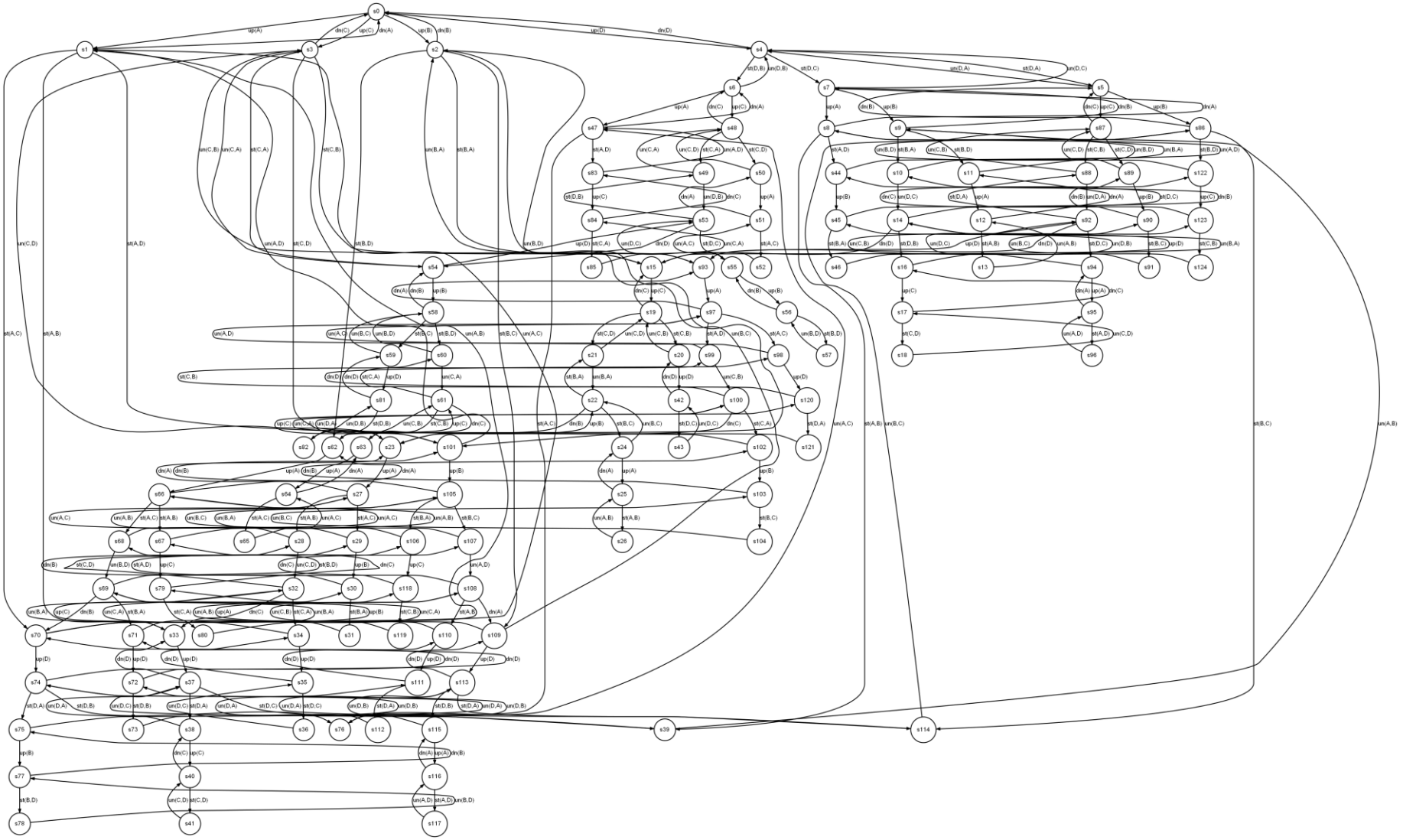
# Let's formalize!

# BW4: Achievable States



jonkv@ida

- Consider physically achievable states in the blocks world, size 4:



# BW4: Ground Atoms



jonkv@ida

- 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)**

# BW4: A Pattern



jonkv@ida

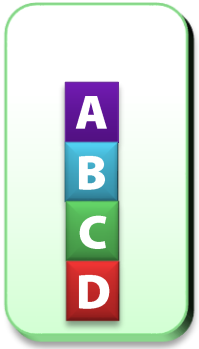
- Example: **only** consider 5 **ground facts** related to **block A**
  - **"Pattern"**: $p$={**(on A B), (on A C), (on A D), (clear A), (ontable A)**}
  - **Initial state:**


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**"

# BW4: Transformed Actions



jonkv@ida

- Pattern $p$={**(on A B), (on A C), (on A D), (clear A), (ontable A)**}
  - **Example action**: (unstack A B)
    - **Before transformation:**
      :precondition (and (handempty) **(clear A) (on A B)**)
      :effect (and (not (handempty)) (holding A) **(not (clear A))** (clear B) **(not (on A B))**)
    - **After transformation:**
      :precondition (and (clear A) (on A B))
      :effect (and (not (clear A)) (not (on A B)))

Loses **some** preconditions and effects

Let's call this action
$a \cap p$
(not a set, but "*restricted to p*")

  - **Example action**: (unstack C D)
    - **Before transformation:**
      :precondition (and (handempty) (clear C) (on C D))
      :effect (and (not (handempty)) (holding C) (not (clear C)) (clear D) (not (on C D)))
    - **After transformation:**
      :precondition (and)
      :effect (and)

Loses **all** preconditions and effects ➔ never used!

# PDB Heuristics: Patterns, Abstract States



jonkv@ida

- 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)

≈

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

≈

**(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)

≈

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

≈

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

represented by a single abstract state

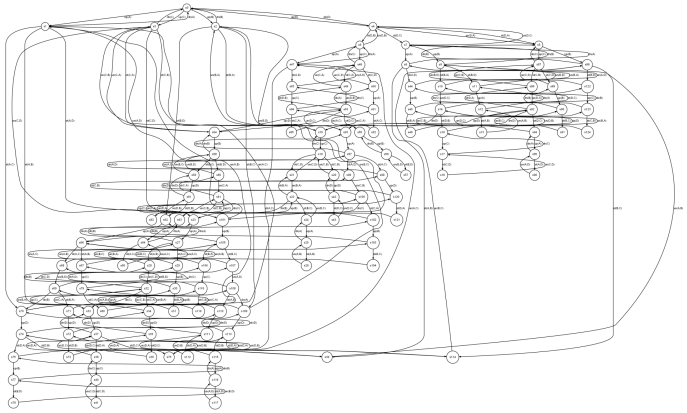
**(clear A)**
**(ontable A)**

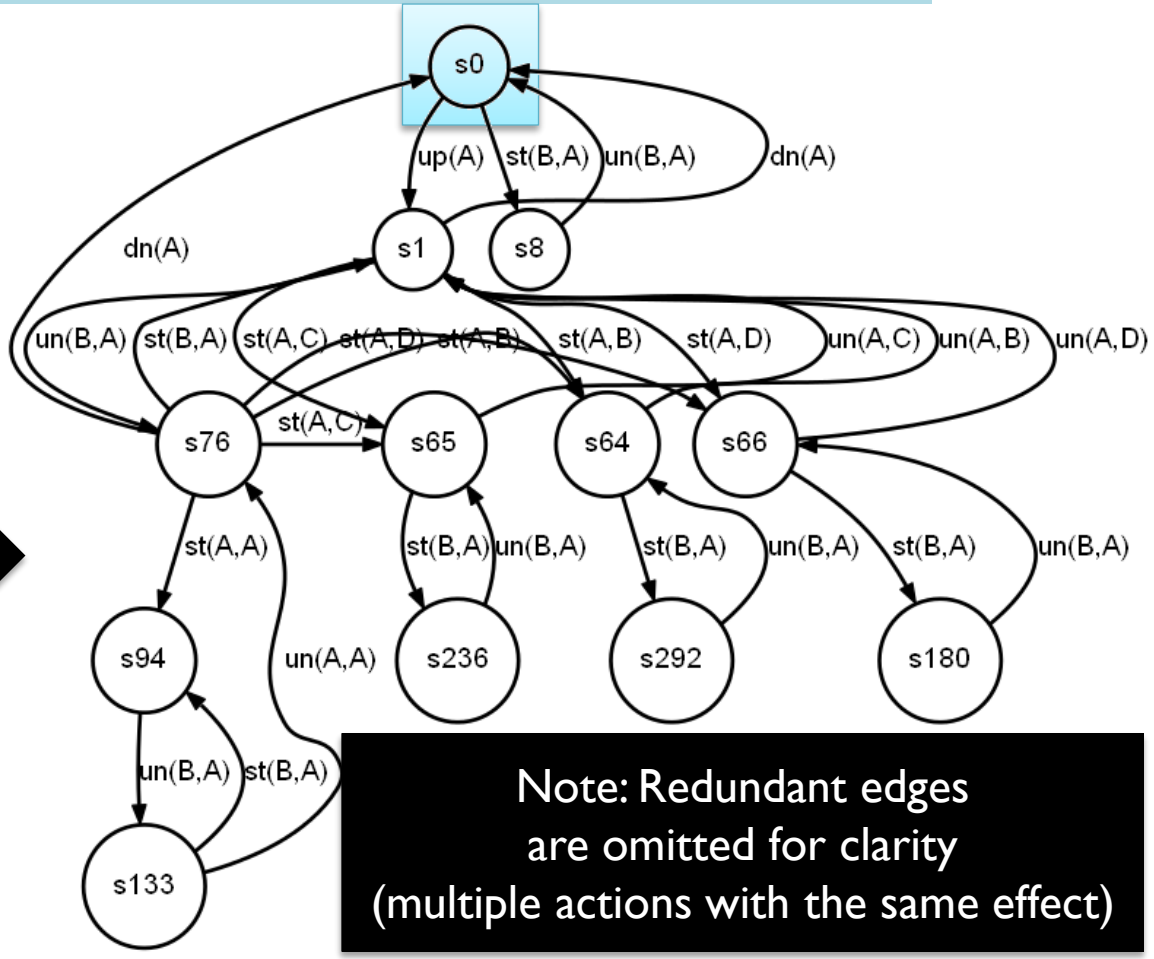
# BW4: Smaller State Transition Graph



jonkv@ida

- **Reachable state transition graph** for the given pattern:
  - **Current state**: Everything on the table, hand empty, all blocks clear
    - Abstract state: s0 = { (ontable A), (clear A) }





# Relaxation!

# Relaxation! (1)



jonkv@ida

- Why is this a relaxation?
  - Step 1: In $s$, we can execute $a_i$ ➔ In $s \cap p$, we can execute $a_i \cap p$

$s$

contains precond($a_i$)

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

$s \cap p$

contains precond($a_i \cap p$)

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

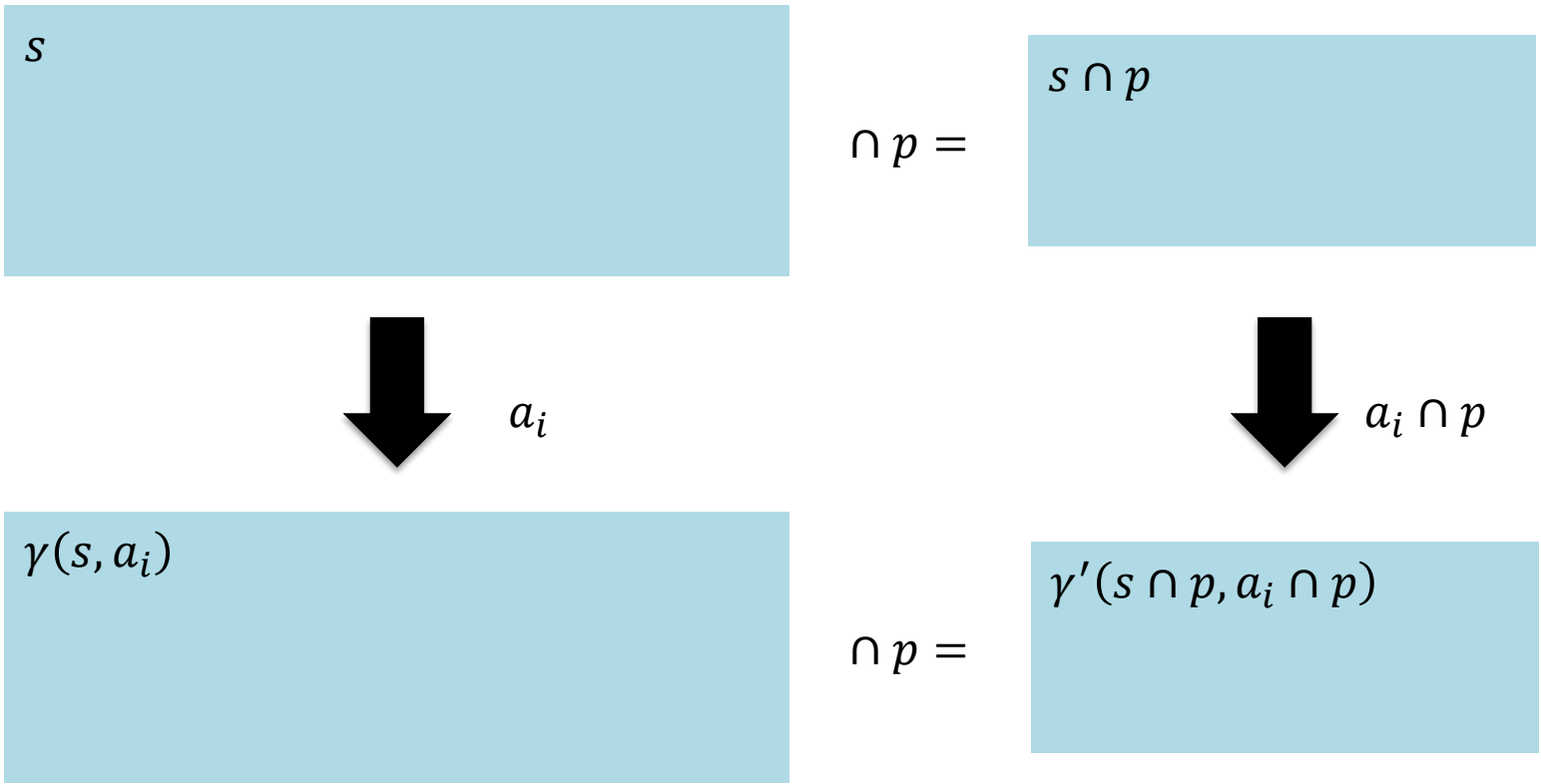
# Relaxation! (2)



jonkv@ida

- Relaxation 2:
  - If $\gamma'$ is the state transition function for transformed actions/states, then:

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




**So: Executable action sequences are preserved**

# Relaxation! (3)



jonkv@ida

- Relaxation 3:
  - If $g \subseteq s$, then $g \cap \mathrm{p} \subseteq s \cap \mathrm{p}$


  - So: **Solutions are preserved** (but new solutions may arise)

# State Representation for PDBs

# PDB Heuristics: State Variables



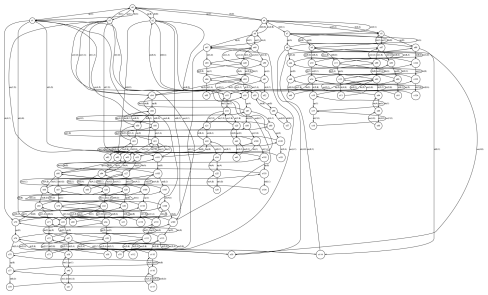
jonkv@ida

- 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 Heuristics: State Variables (2)



jonkv@ida



- Repetition: Blocks world with 4 blocks
  - **536,870,912 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 a tower
      - **(on B A)** – Below B
      - **(on C A)** – Below C
      - **(on D A)** – Below D

    - ➔ Remove those facts, introduce state variable

      **aboveA ∈ { gripper, nothing, B, C, D }**

# PDB Heuristics: State Variables (3)



jonkv@ida

- Example, continued
  - 536,870,912 states (reachable and unreachable) in predicate representation
  - 20,000 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**

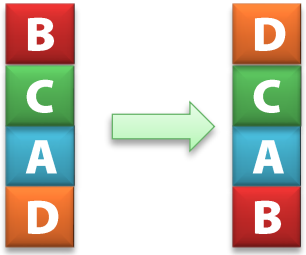
**Also useful when choosing facts: Ignore where A is, care about where B is**

# PDB Heuristics: Rewriting the Problem



jonkv@ida

- **Rewriting** works as before
  - Suppose the pattern is **{ aboveB, aboveD, posB, posD }**




  - **Rewrite** the goal
    - **Original:** { aboveB=A, aboveA=C, aboveC=D, 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 aboveB = 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 Heuristics: State Space Size

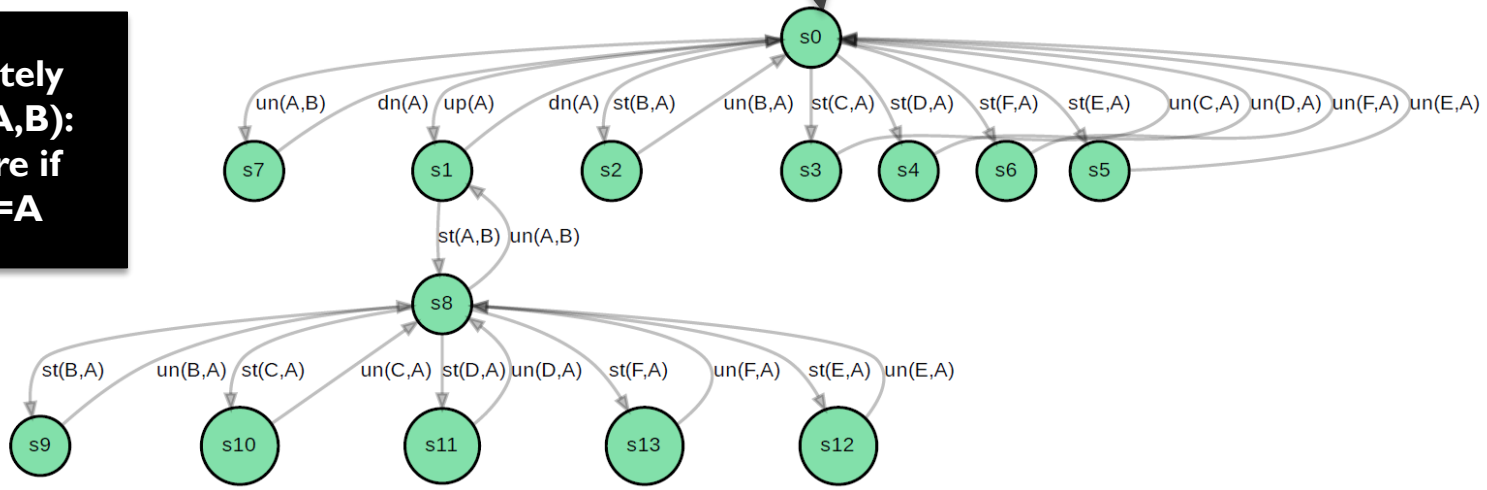


jonkv@ida

- **Abstract** states reachable from "all on table", **by pattern p**…

| Blocks | All vars | p={aboveA} | p={aboveA,posA} | P={aboveA,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**

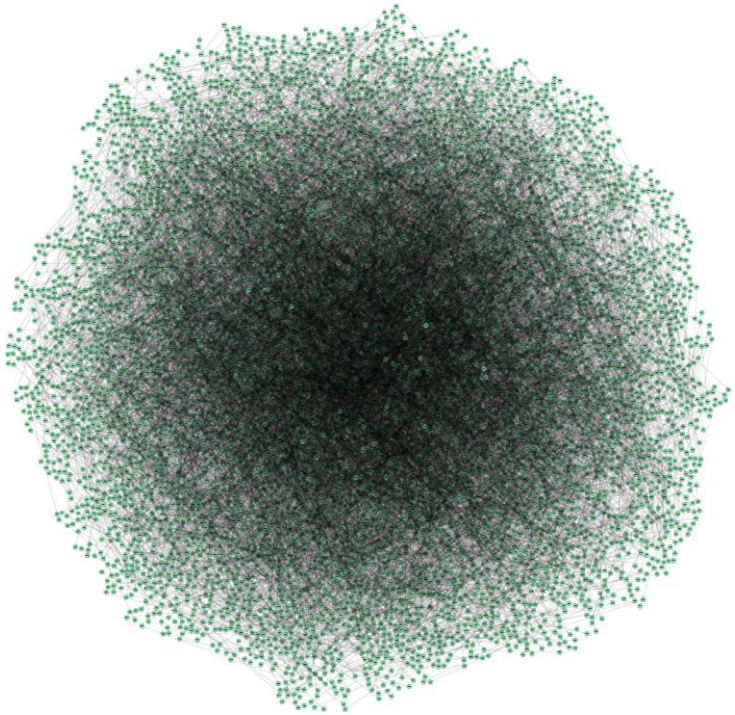


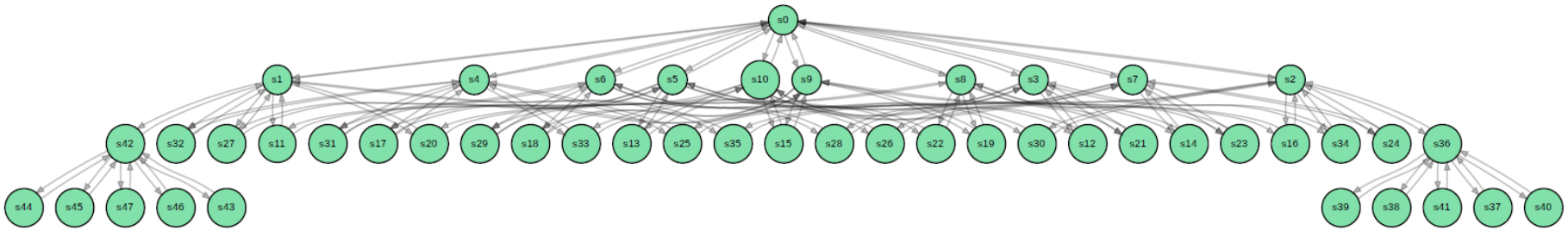
# PDB Heuristics: State Space Size (2)



jonkv@ida

- For 6 blocks, originally 7057 reachable states
  - Pattern **{aboveA,aboveB}** ➔ 48 reachable states





# Computing PDB Heuristics

# PDB Computation: Main Idea



jonkv@ida

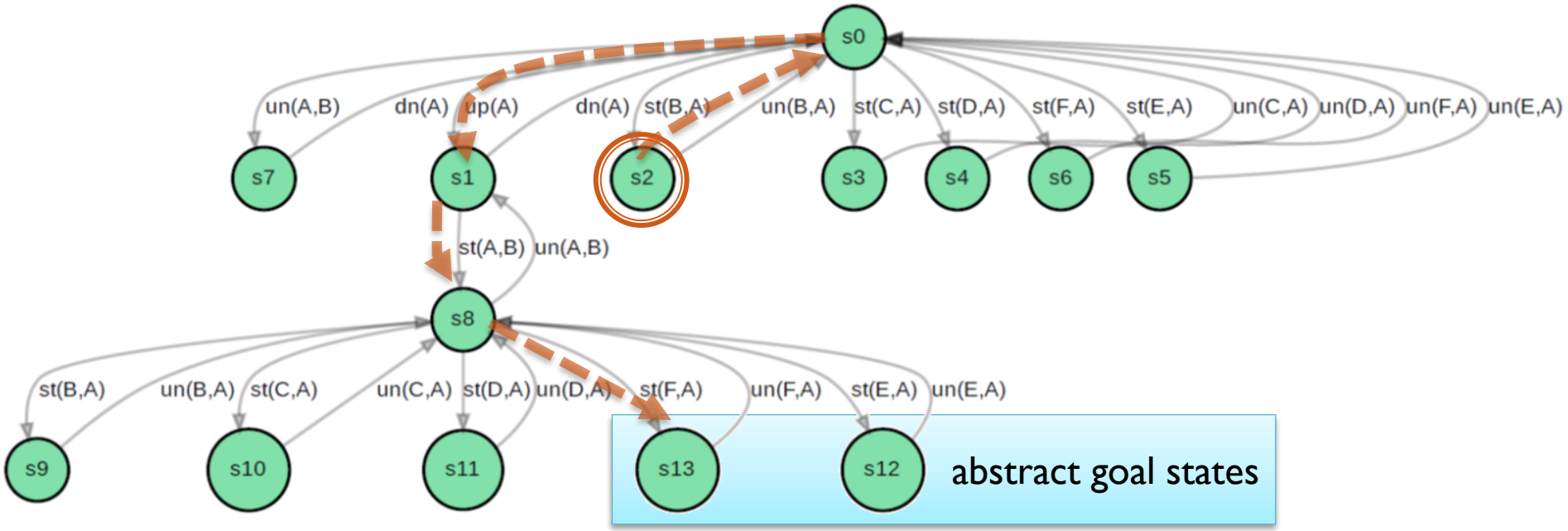
- To calculate $h(s)$ for a new state $s$:
  - Example: 6 blocks, pattern **{aboveA,posA}**
  - Convert $s$ to *abstract* state:

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

↓

aboveA=B
posA=on-table

  - Find **optimal** path to abstract goal state – in a much smaller search space
    - Current abstract state is $s_2$:

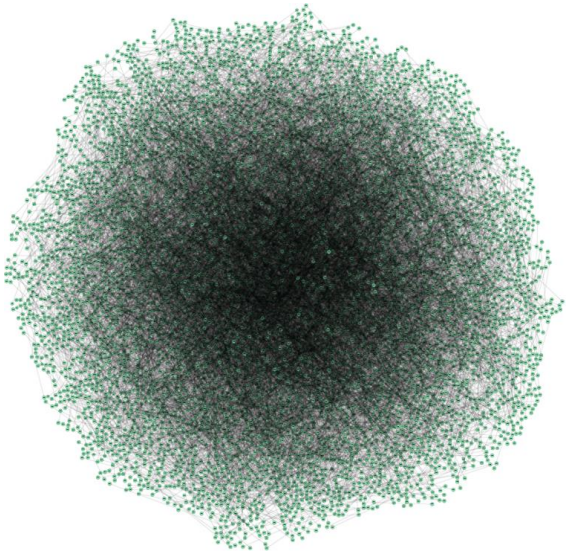


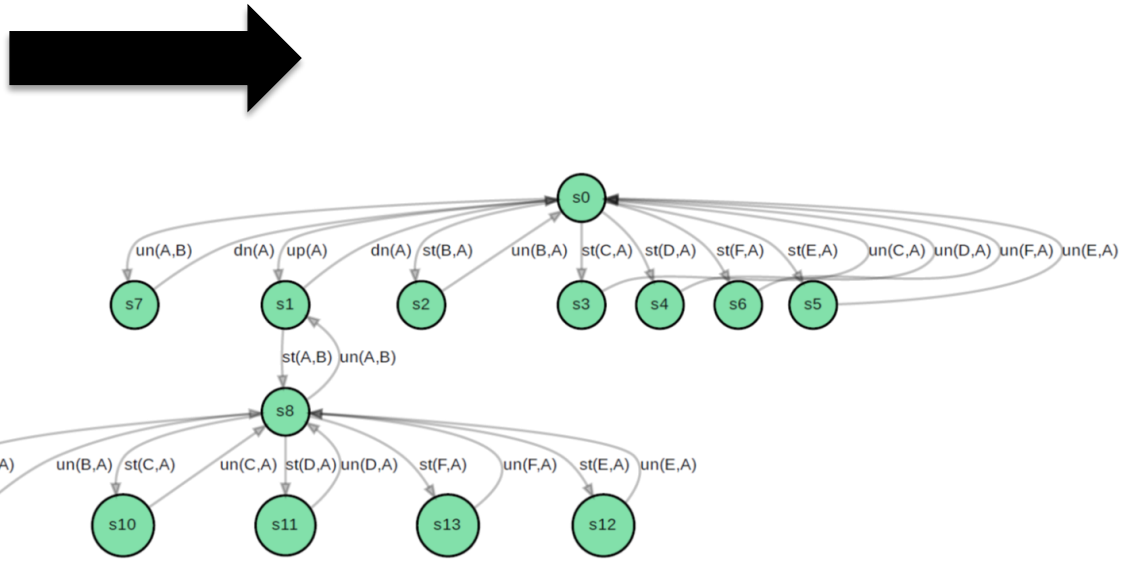
# Tweak 1: Caching



jonkv@ida

- 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!



jonkv@ida

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

**Second main idea!**

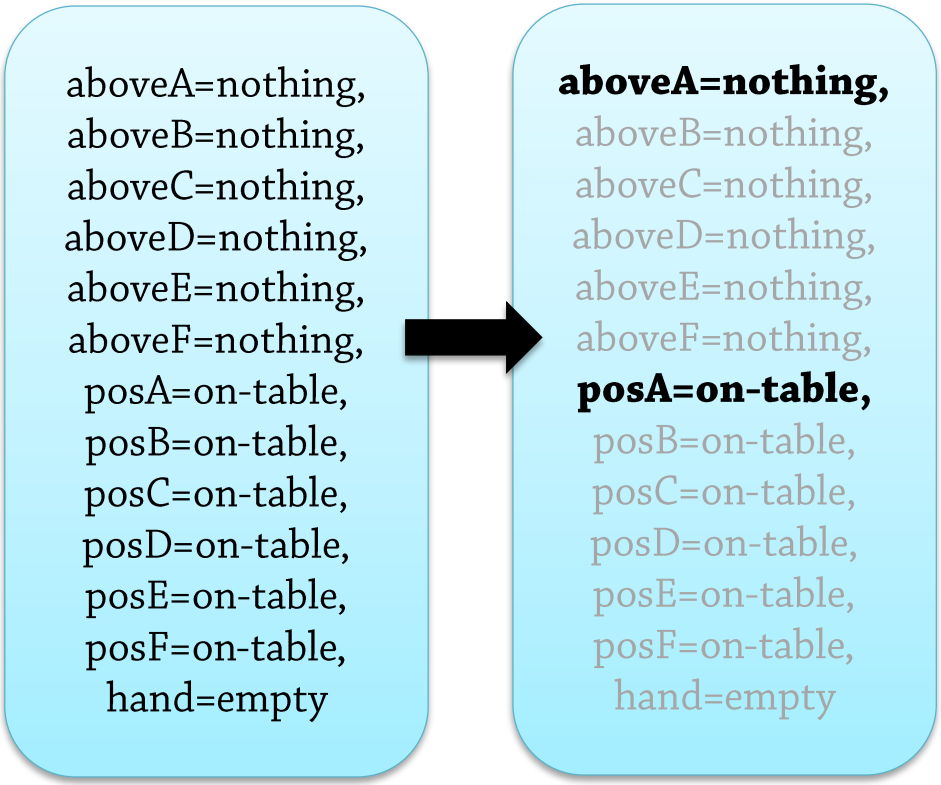
# Database Creation

# Preprocessing (1)



jonkv@ida

- **Preprocessing** step 1 (6 blocks, pattern **{aboveA, posA}**)
  - Create the initial *abstract* state…

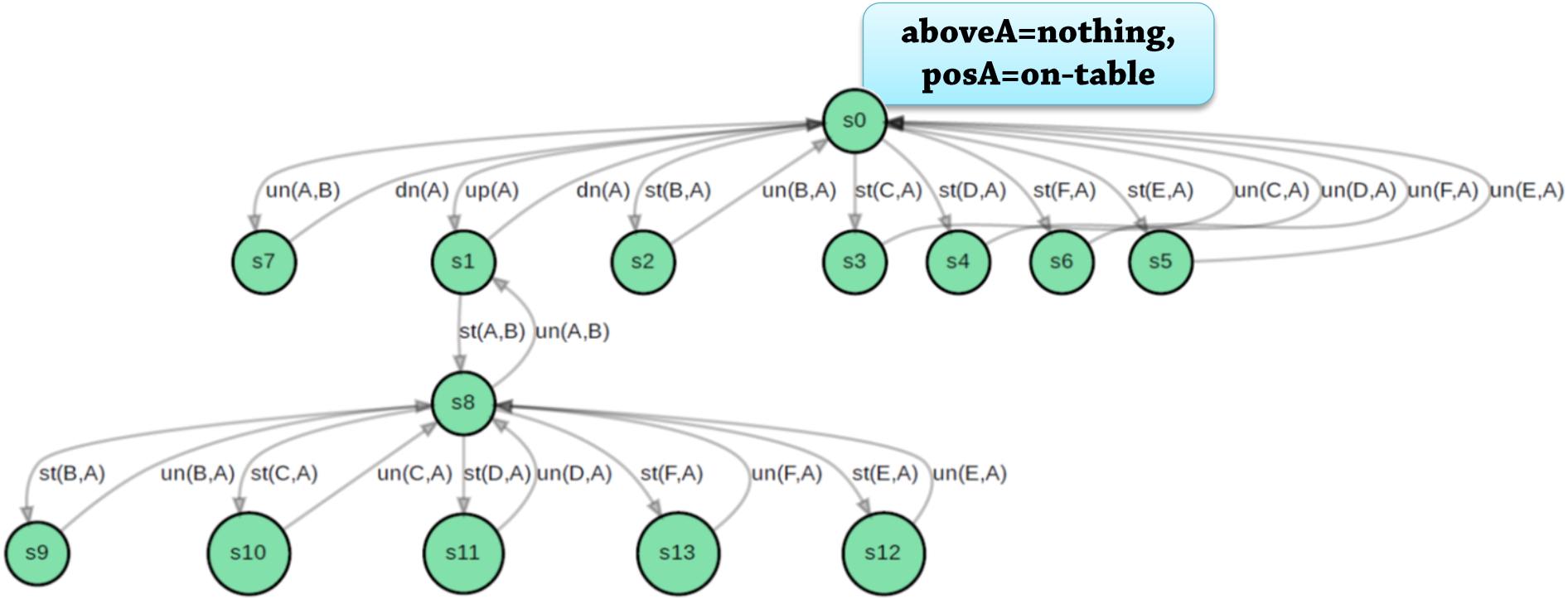


# Preprocessing (2)



jonkv@ida

- **Preprocessing** step 2
  - Find **all reachable** abstract states
    - Start in the abstract initial state (s0 below)
    - Apply all applicable actions to all states you find (DFS, BFS, …)
    - Small, therefore *fast*

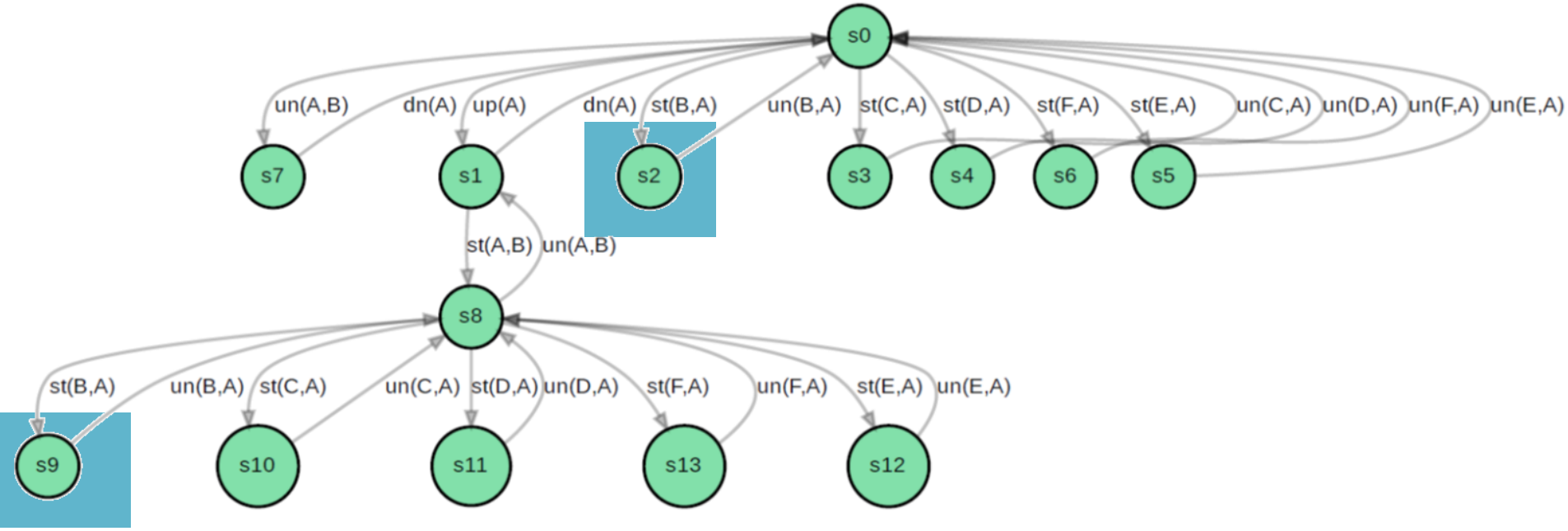


# Preprocessing (3)



jonkv@ida

- 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

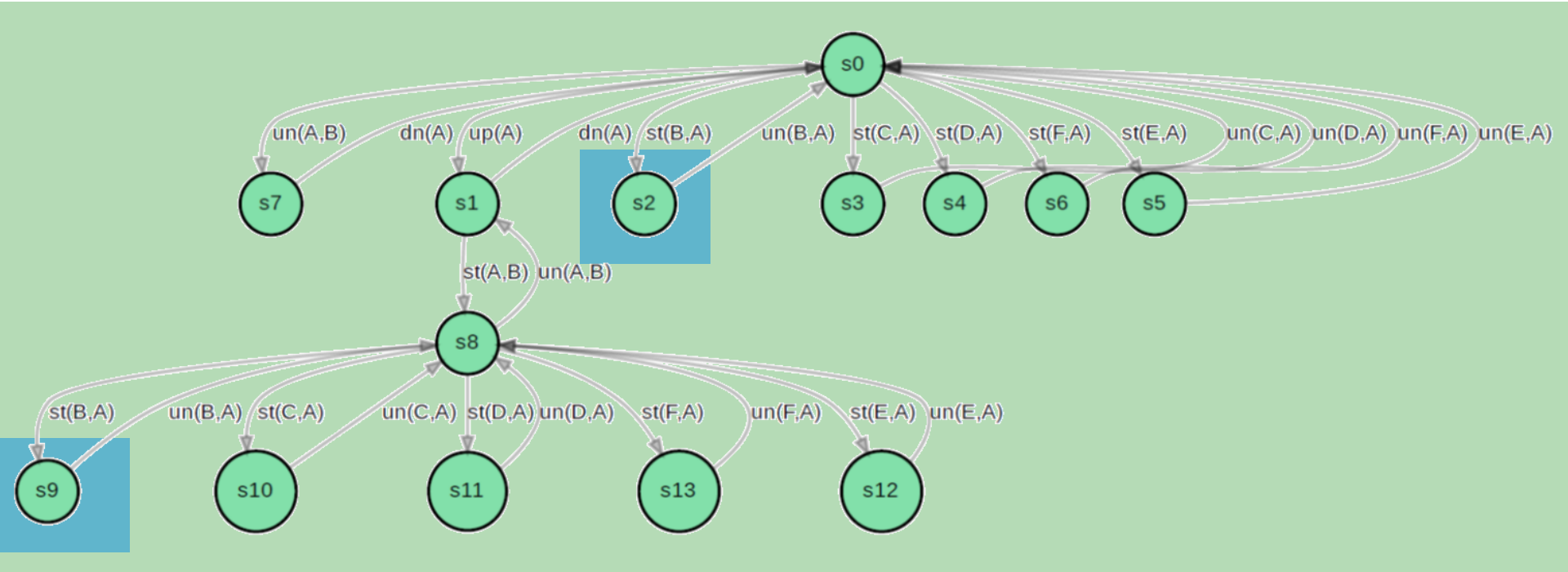


# Preprocessing (4)



jonkv@ida

- 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**

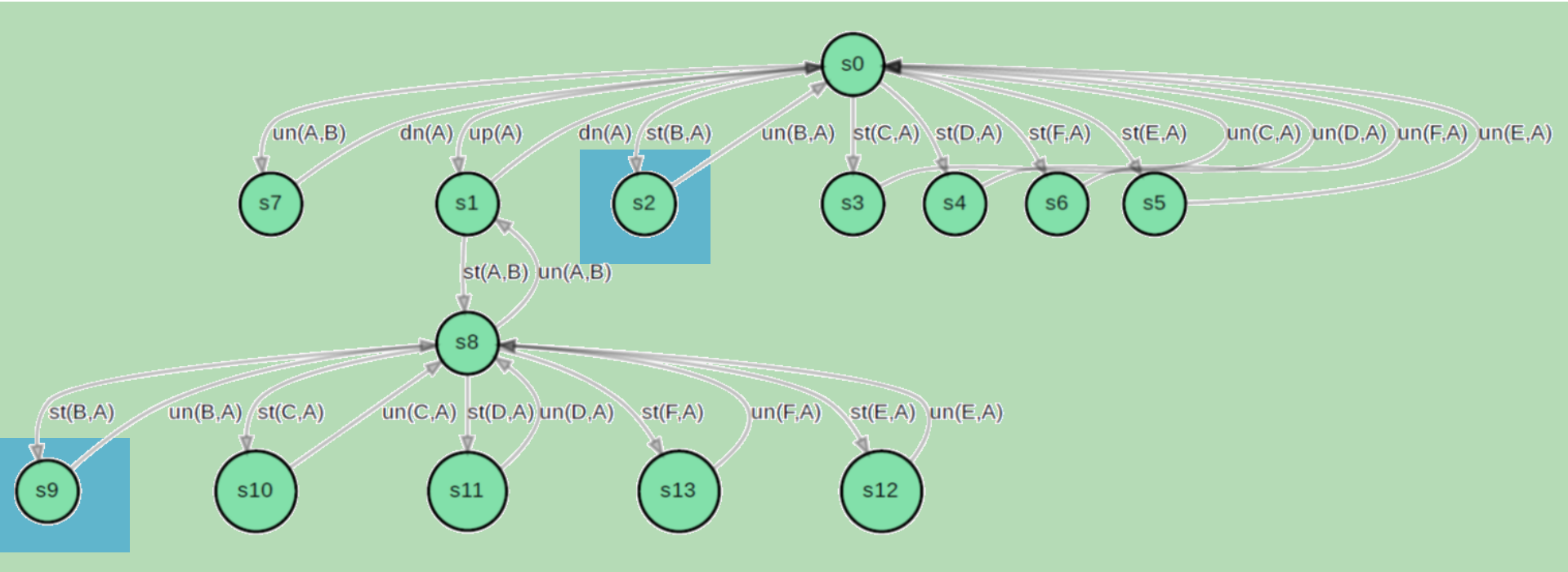


# Preprocessing (5)



jonkv@ida

- 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 (s2, s9)
    - Edges followed backward (in explicitly computed graph, no $\gamma^{-1}$ necessary)



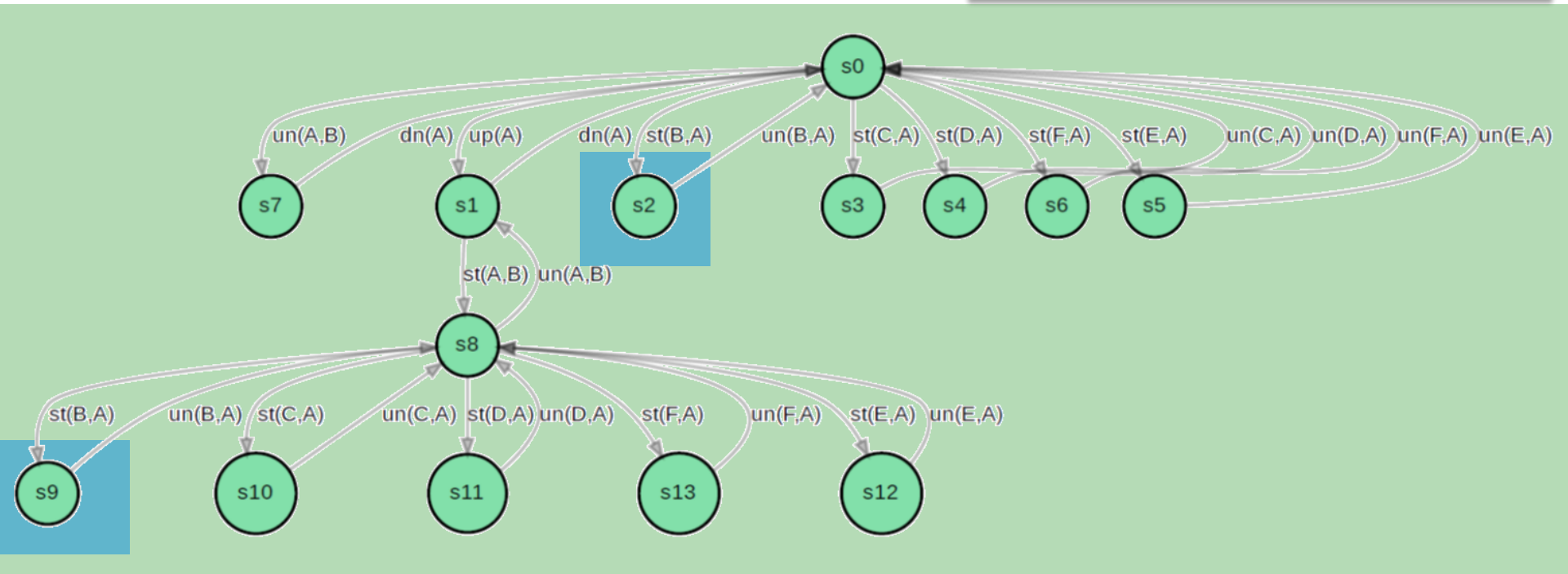
# Preprocessing (6)



jonkv@ida

- Example: Priority queue contains <s2/cost=0,s9/cost=0>
  - Pick s2
    - A ”successor” is s0, reached backwards by stack(B,A) of cost 2;
    - s0 inserted in queue with cost 2
  - Priority queue contains <s9/c=0, s0/cost=2>, pick s9
    - A ”successor” is s8, …

Assuming cost(stack/unstack)=2, cost(pickup/putdown)=1



# PDB Heuristics: Databases



jonkv@ida

- 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

| Abstract state | Cost |
| --- | --- |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

# Using PDB Heuristics during Search

# PDB Heuristics in Forward Search (1)



jonkv@ida

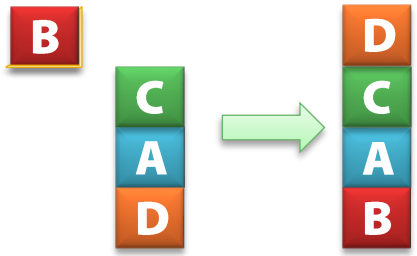
- Step 1: **Automatically** generate a pattern
  - A selection of state variables to consider
  - Choosing a **good** pattern is a **difficult problem**!
    - Different approaches exist…
- Step 2: Calculate the pattern database
  - As already discussed

# PDB Heuristics in Forward Search (2)



jonkv@ida

- Step 3: Forward search in the **original problem**
  - For each new successor state $s_1$, calculate heuristic value $h_{pdb}(s_1)$
    - Example: $s_1$ = { aboveD = A, aboveA = C, aboveC = nothing, aboveB = gripper, posA = other, posB = other, posC = other, posD = on-table, hand = full }




    - Convert this to an *abstract* state
      - Example: $s_1'$ = { aboveB = gripper, aboveD = A, posB = other, posD = on-table }
    - Use the database to quickly look up $h_{pdb}(s_1)$ = the cost of reaching the nearest abstract goal from $s_1'$

aboveB = gripper, aboveD = A, posB = other, posD = on-table ➔ cost *n1*
aboveB = gripper, aboveD = A, posB = other, posD = other ➔ cost *n2*
…

# How informative can PDBs be?

# Computation Time (1)



jonkv@ida

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

**Exponential in the number of state variables in the pattern…**

**Possible state variables: Linear in problem size (length of PDDL file, …)**
**Select all ➔ PDB computation exponential in problem size**

# Computation Time (2)



jonkv@ida

**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(problem size)**

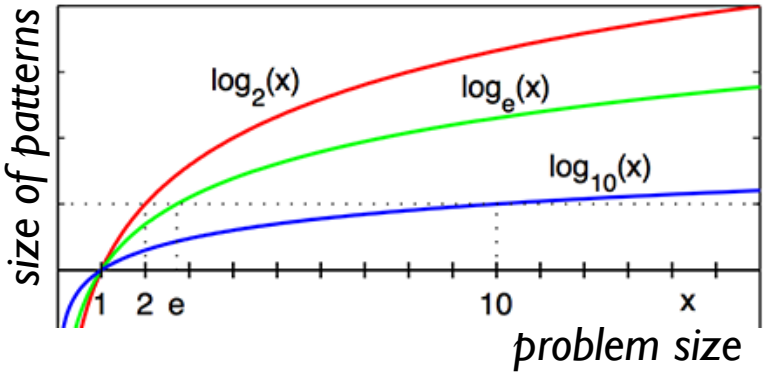
**Computation time: $O(2^{\log problem\ size})$, which is polynomial in problem size**

# Asymptotic Accuracy: Single PDB Heuristic



jonkv@ida

- **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



jonkv@ida

- 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:**

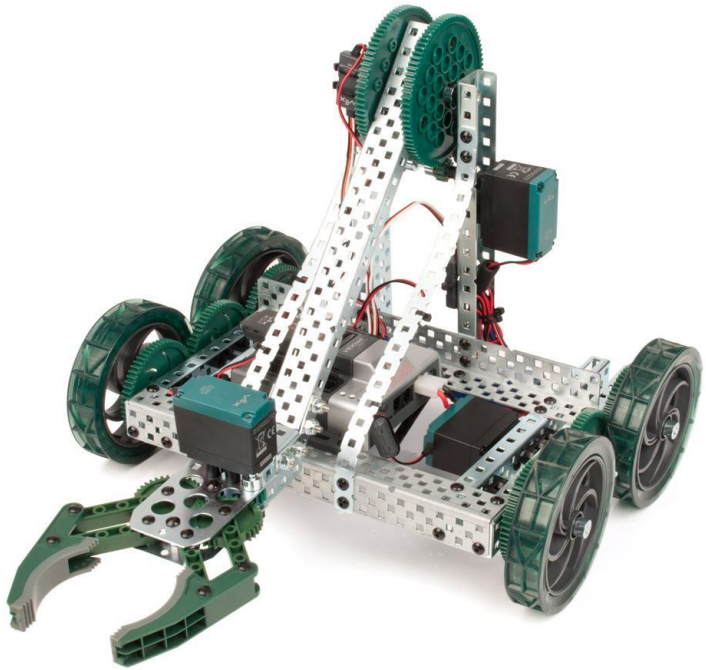
**loc(ball$_k$)** ∈ { room1, room2, gripper1, gripper2 }

**loc-robot** ∈ { room1, room2 }

*$2 * 4^n$ states, some unreachable – which ones?*

**Some possible patterns for k ≥ 1 balls:**

| | |
|---|---|
| { loc(ball$_1$) } | ➔ 4 abstract states |
| { loc(ball$_1$), loc-robot } | ➔ 8 abstract states |
| { loc(ball$_i$) \| i ≤ k} | ➔ $4^k$ abstract states |
| { loc(ball$_i$) \| i ≤ log(k)} | ➔ $4^{\log(k)}$ abstract states |



# Multiple Patterns:

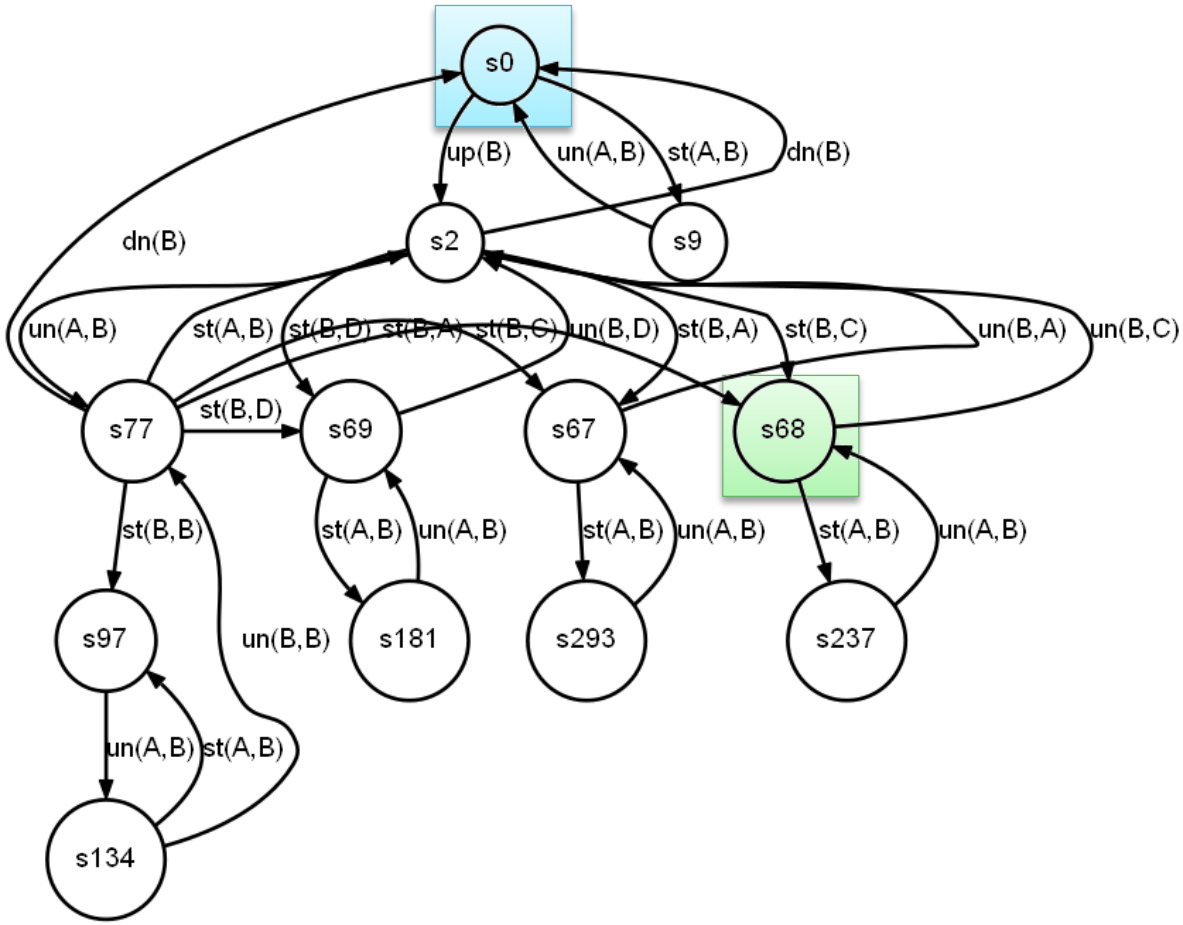
# More information,
# Same pattern size

# BW4: Subproblem 2



jonkv@ida

- Subproblem 2: Some facts related to **B**
  - **Current state**: Everything on the table, hand empty, all blocks clear
    - Abstract state: **{ (ontable B), (clear B) }**
  - **Goal state**:
    A on B on C on D
    - Abstract goal:
      **{ (on B C) }**
  - Find a path,
    compute its cost

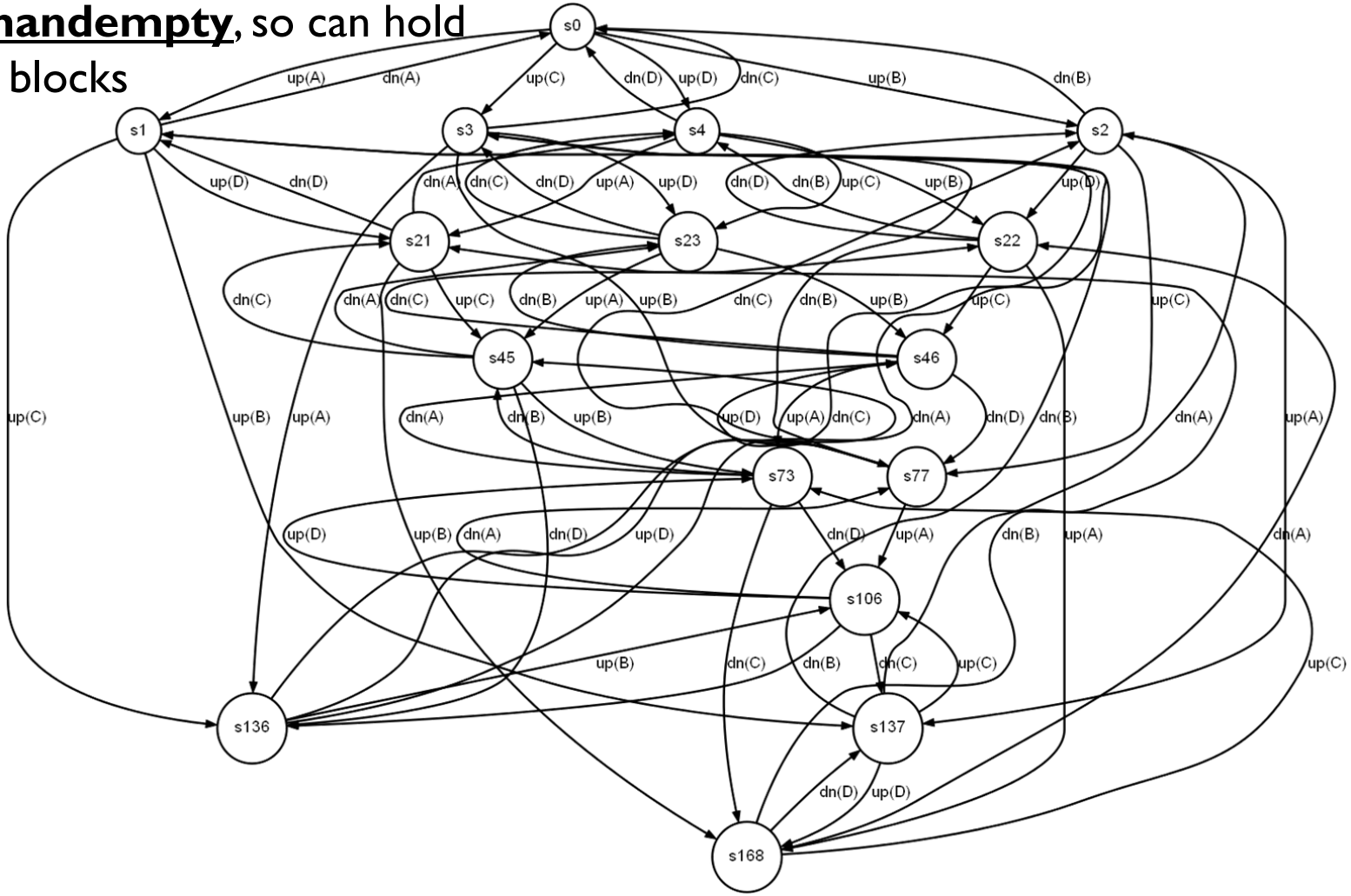


# BW4: Subproblem 3



jonkv@ida

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



# Improving PDBs



jonkv@ida

- 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: | Pattern 2, aboveC: | Sum: |
|---|---|---|
| I need stack(A,B), cost 2 | I need stack(B,C), cost 2 | 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

# Additive PDB Heuristics (1)



jonkv@ida

- 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

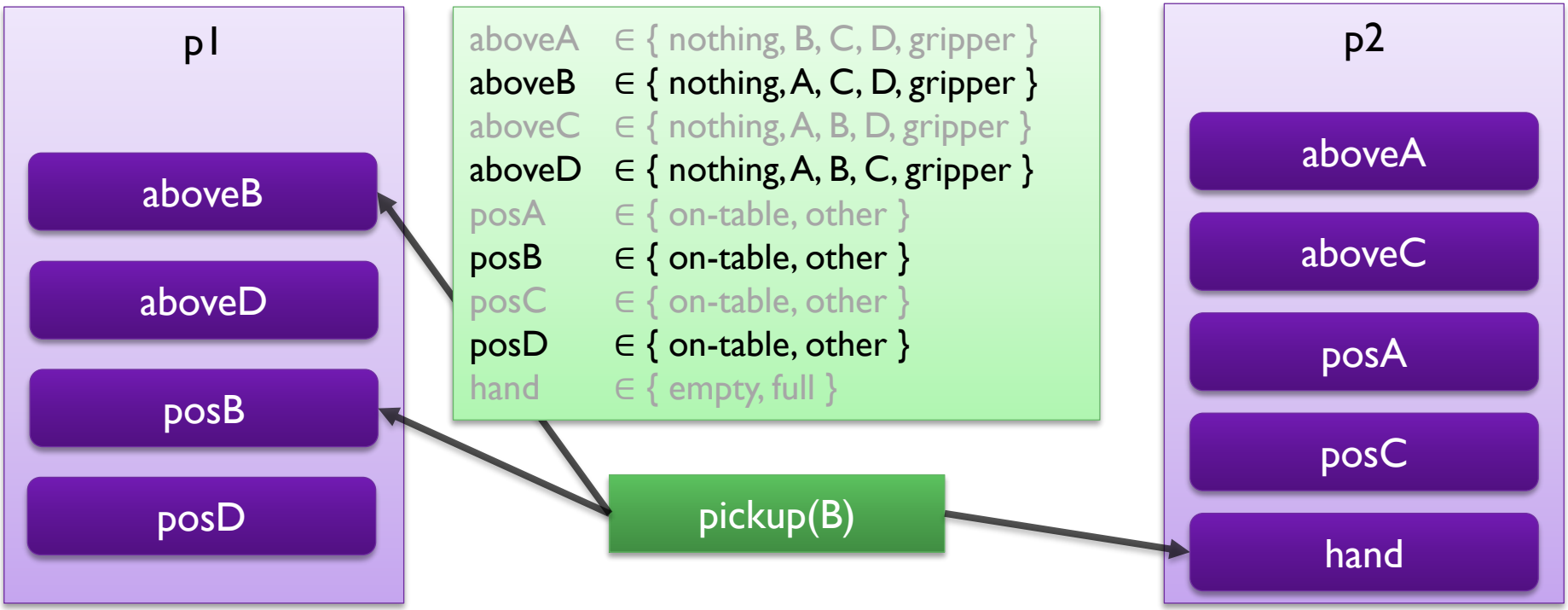
- → **Additive** pattern database heuristics

# Additive PDB Heuristics (2)



jonkv@ida

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



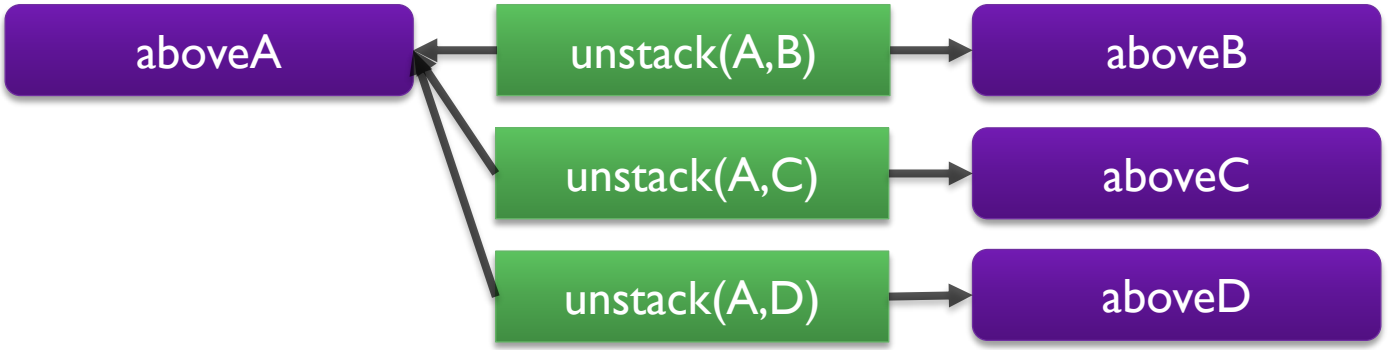

**One potential problem:**
**Both patterns could use pickup(B) in their optimal solutions**
**➔ sum counts this twice! This is what we're trying to avoid…**

# Additive PDB Heuristics (3)



jonkv@ida

- BW: Is p1={aboveA}, p2={aboveB} additive?
  - No: unstack(A,B) affects {aboveB} in p1, {aboveA} in p2
  - True for *all* combinations of above*X*




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

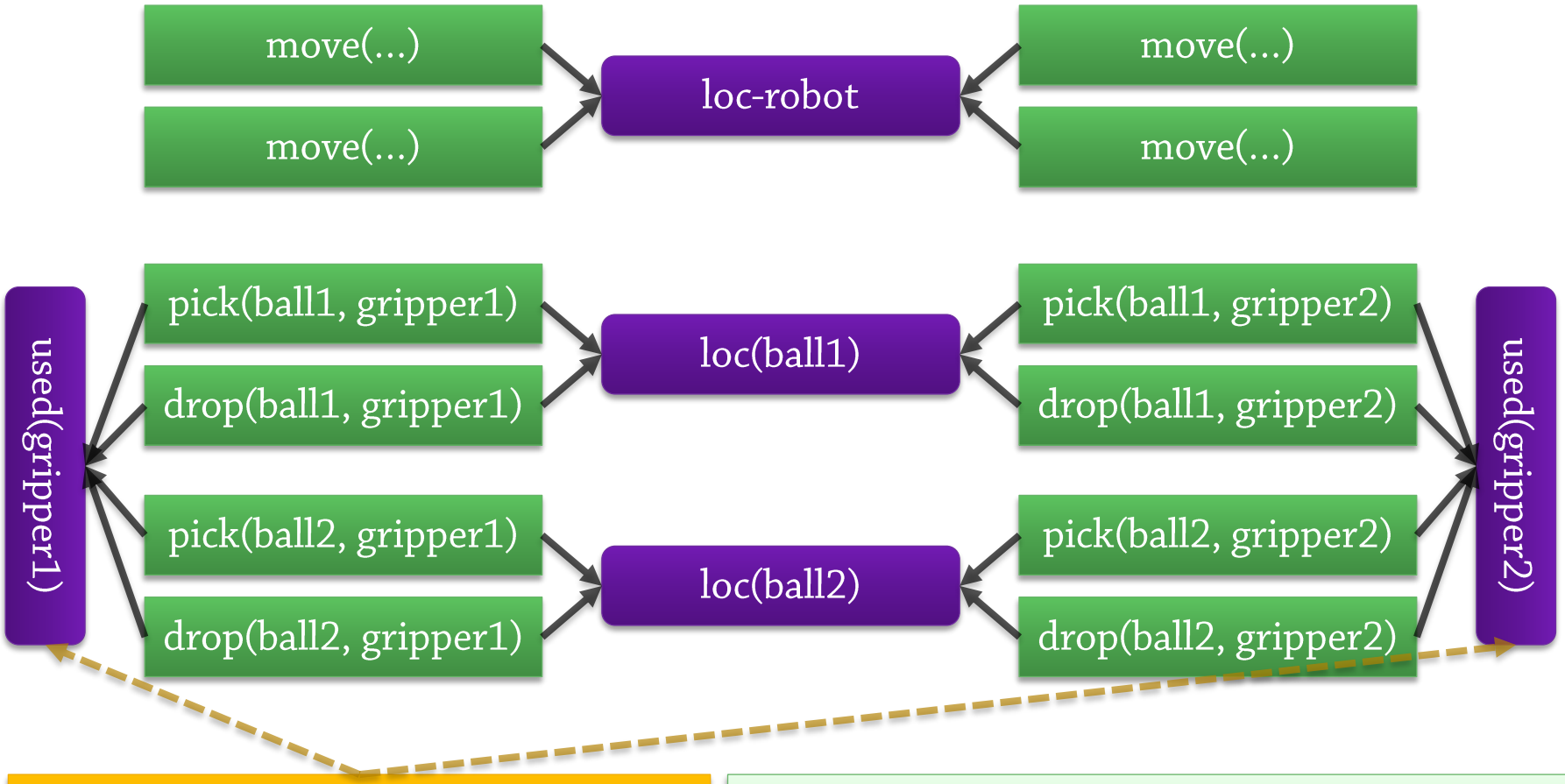
**This formulation of the Blocks World is "connected in the wrong way" for this approach to work well**

# Additive PDB Heuristics (4)



jonkv@ida

- "Separating" patterns in the Gripper domain:




**Are these a problem?**

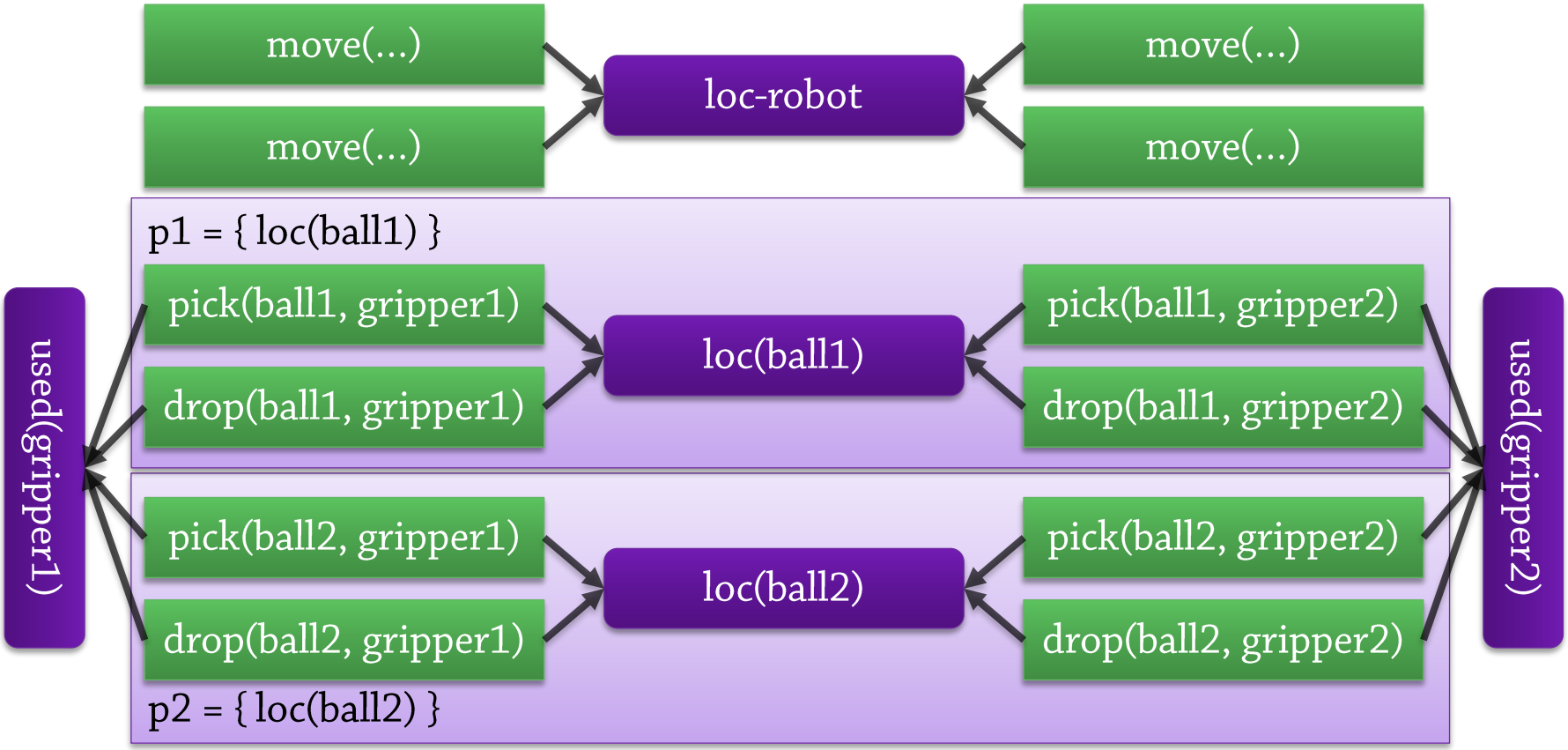
$loc(ball_k)$ $\in$ { room1, room2, gripper1, gripper2 }
loc-robot $\in$ { room1, room2 }
$used(gripper_k)$ $\in$ { true, false }

# Additive PDB Heuristics (5)



jonkv@ida

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



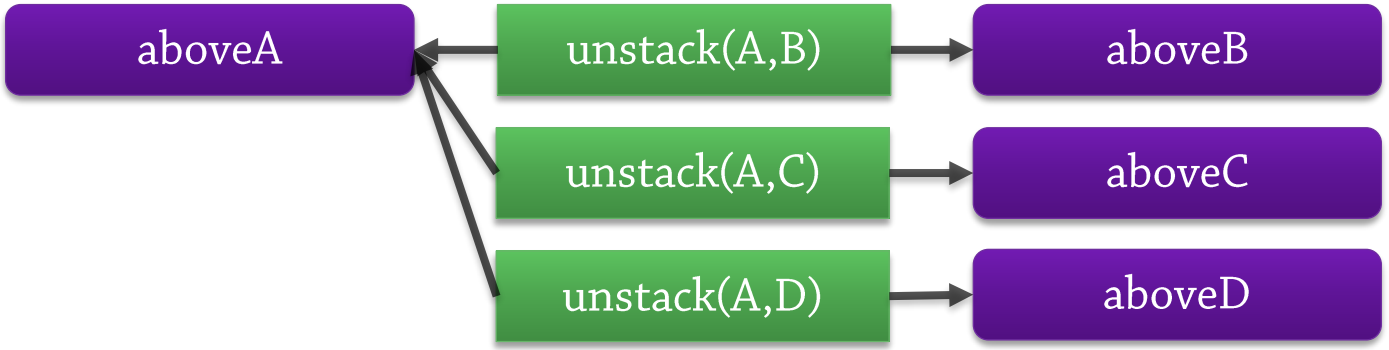

For each pattern we chose one **variable**
Then we have to include **all actions** affecting it
The other variables those actions affect [used()] don't have to be part of *any* pattern!

# Additive PDB Heuristics (6)

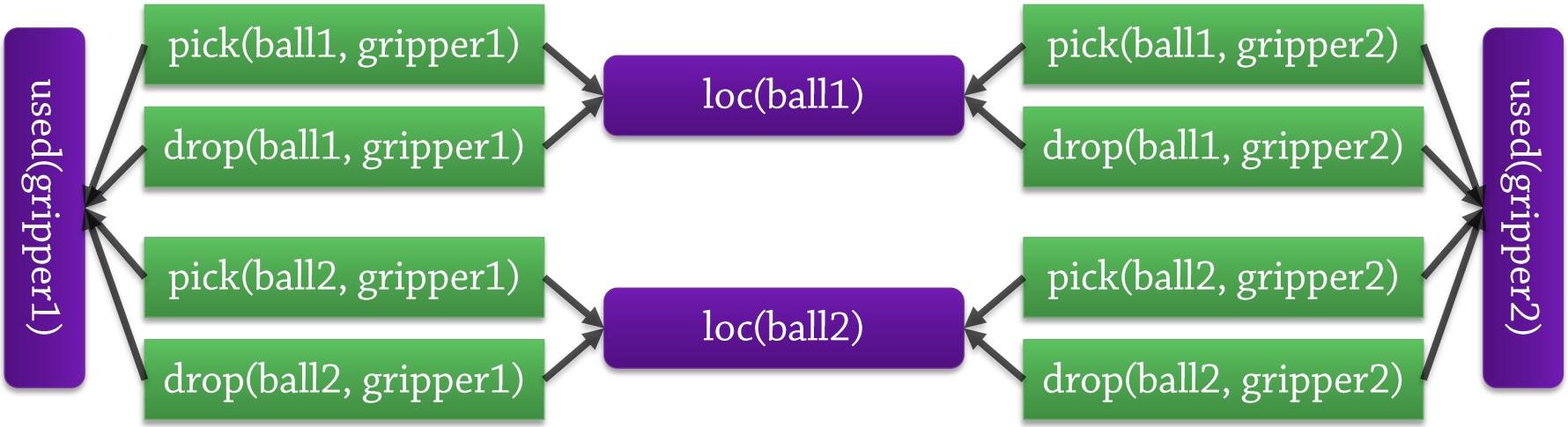


jonkv@ida

- Notice the difference in structure!




BW: *Every* pair of above*X* facts has a *direct connection* through an action




Gripper: *No* pair of loc() facts has a *direct connection* through an action

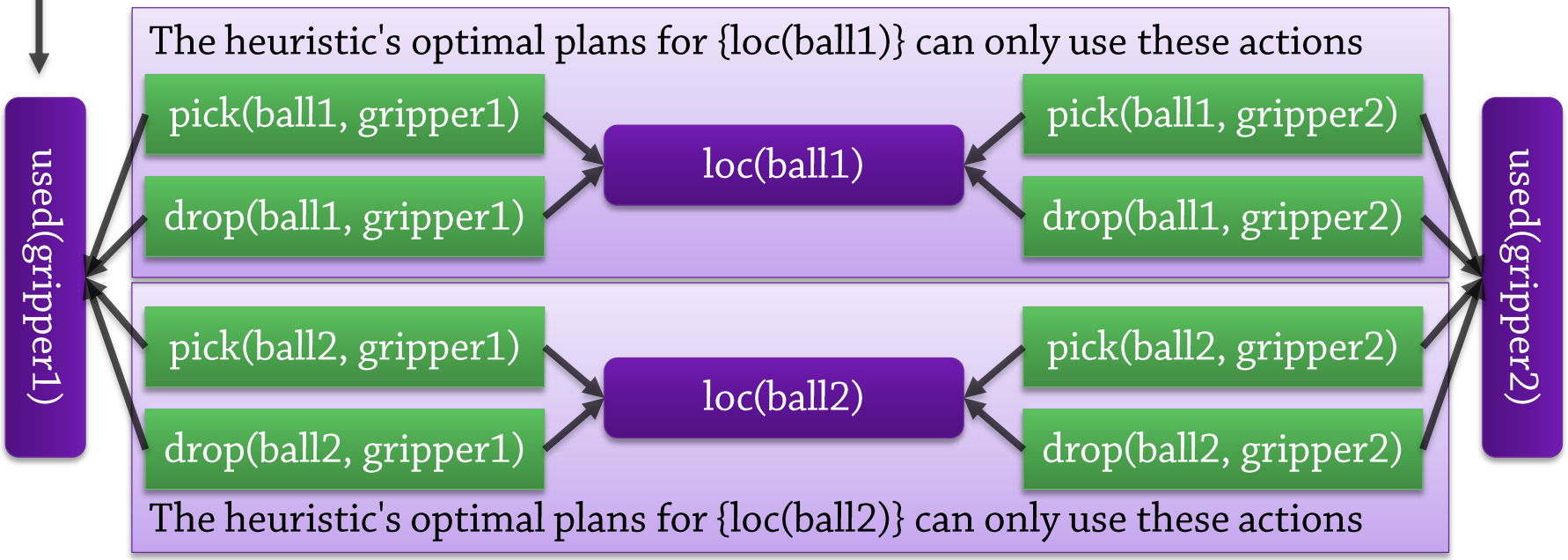
# Additive PDB Heuristics (7)



jonkv@ida

- 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 (8)



jonkv@ida

- 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

→ The costs are additive for multiple patterns
→ Adding costs from multiple heuristics yields an *admissible* heuristic!

# Additive PDB Heuristics (9)



jonkv@ida

- 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

Max ➔
admissible heuristic $h^3_{pdb}(s) = \max(h^1_{pdb}(s), h^2_{pdb}(s))$

Sum ➔
admissible heuristic $h^1_{pdb}(s)$

p1 | p2 | p3 | p4

4 patterns satisfying additive constraints

Sum ➔
admissible heuristic $h^2_{pdb}(s)$

p5 | p6 | p7 | p8 | p9

5 patterns satisfying additive constraints

# Additive PDB Heuristics: Accuracy

# Additive PDB Heuristics (10)



jonkv@ida

- **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 (11)





- **In Gripper**:
  - In state $s_n$ there are *n* balls in **room1**, and no balls are carried
  - Additive PDB heuristic $h^{PDB}_{add}(s_n)$:
    - Use a separate pattern for each ball location variable **loc(ball$_k$)**
    - For ***each*** pattern/ball, the optimal PDB cost is 2
      - **pick(ball,room1,gripper1): loc(ball)=room1 ➔ loc(ball)=gripper1**
      - **drop(ball,room2,gripper1): loc(ball)=gripper1 ➔ loc(ball)=room2**
    - $h^{PDB}_{add}(s_n)$ = sum for *n* balls = 2*n*
  - Real cost:
    - Using both grippers:
      - *pick, pick, move(room1,room2),*
        *drop, drop, move(room2,room1)*
    - Repeat n/2 times, total cost $\approx$ 6n/2 = 3n
  - ➔ Asymptotic accuracy 2n/3n = 2/3

# Additive PDB Heuristics (12)



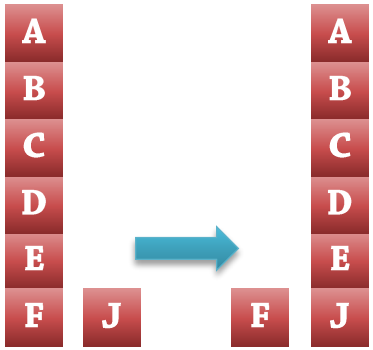
jonkv@ida

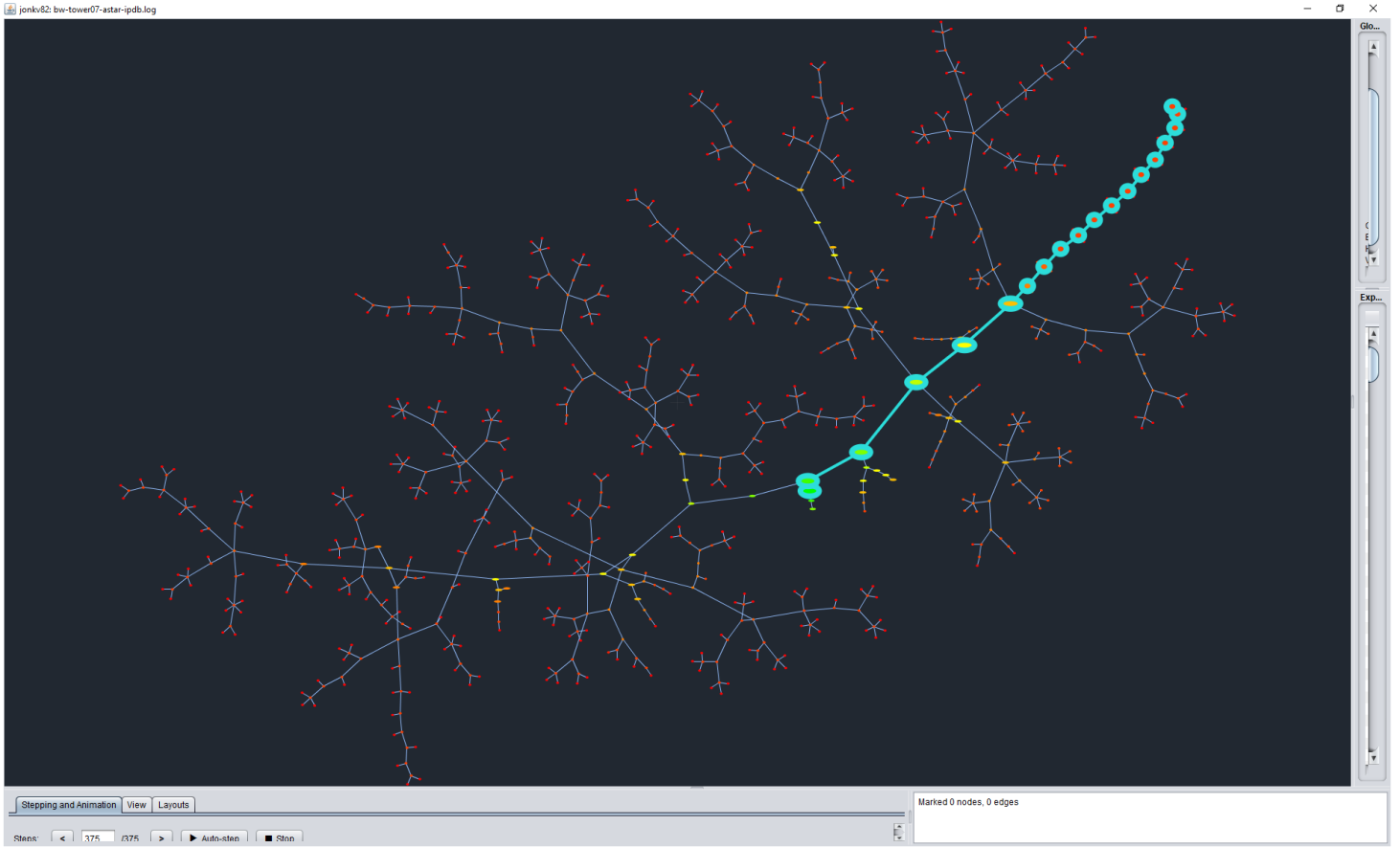
- What quality ***guarantees*** can an ***additive*** PDB heuristic give?
  - For additive PDB heuristics with a single sum, **asymptotic accuracy** (fraction of $h^*$) as problem size approaches infinity:

| | 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…

# 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!**