Linköping University

Automated Planning

The Forward State Space

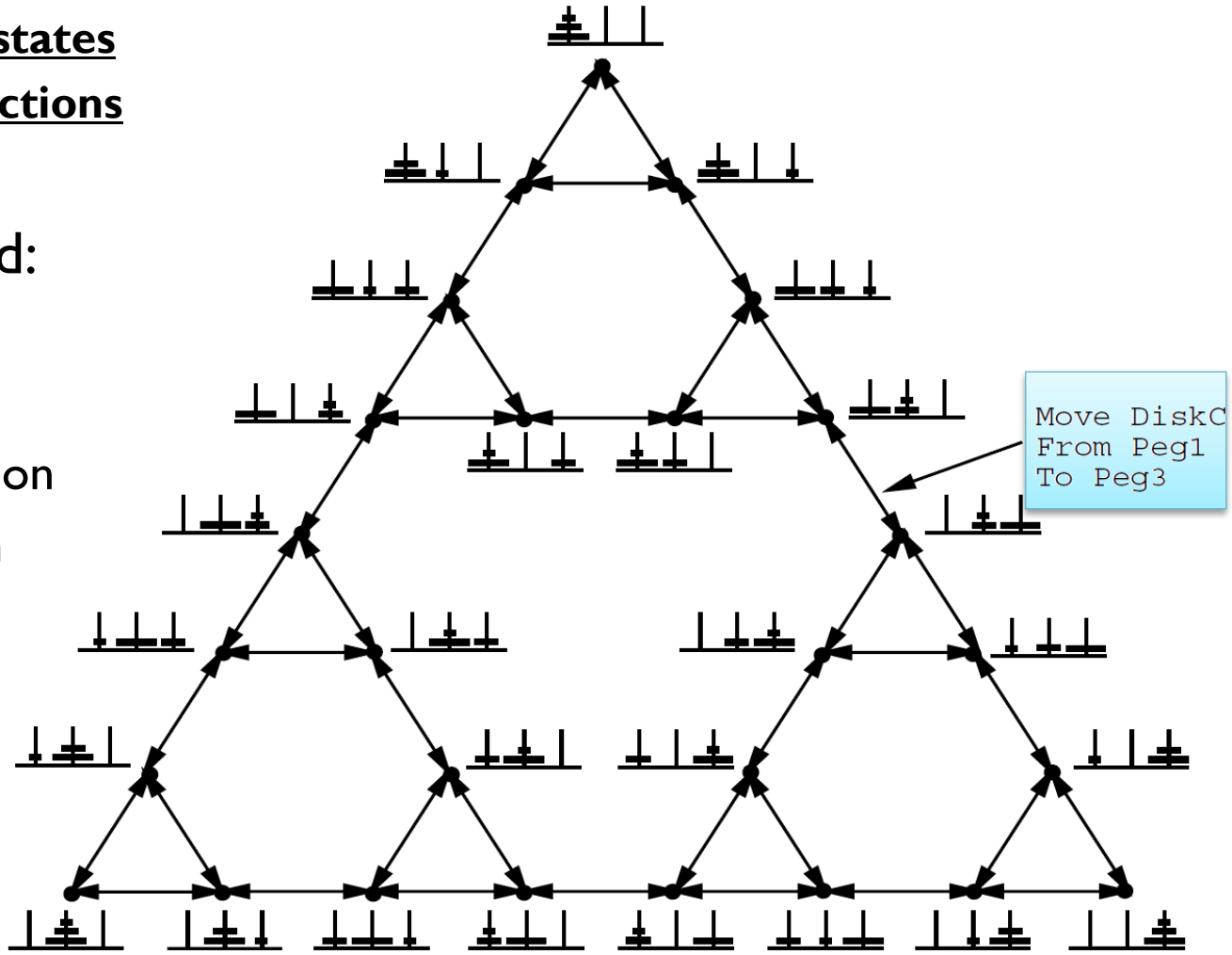
Jonas Kvarnström
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2019

State Space (1)

jonkv@ida

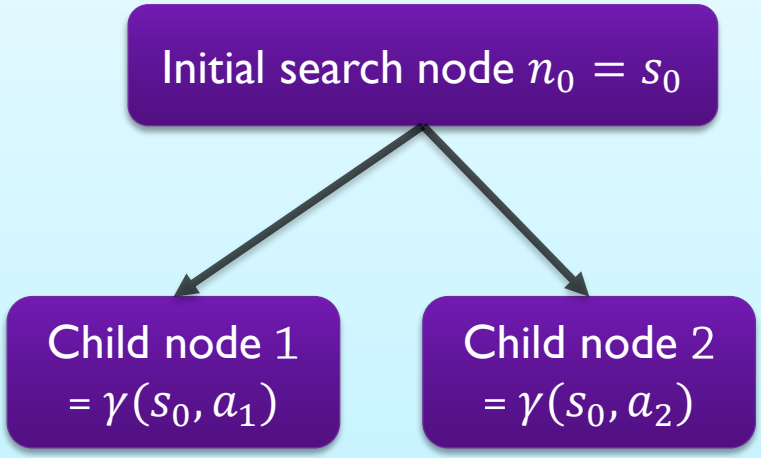
- The **state space**: An "obvious" search space
  - [1] Structure:
    - Nodes represent **states**
    - Edges represent **actions**
- …But we still need:
  - [2] Initial node
  - [3] Branching rule
  - [4] Solution criterion
  - [5] Plan extraction



State Space (2)

jonkv@ida

The **state space** for **forward planning, forward-chaining, progression**



Given a node n:
- I know how to reach the *state* of n
- *If* I can find a path from *n* to a *goal state*, I will be done

2) Initial search node:
Corresponds directly to the initial state

3) Branching rule:
For *every* action a **applicable** in a state s, generate **the state** $\gamma(s, a)$

"Forward", "Progression":
Applying actions in their natural direction

4) Solution criterion: *The state of the node satisfies the goal formula*

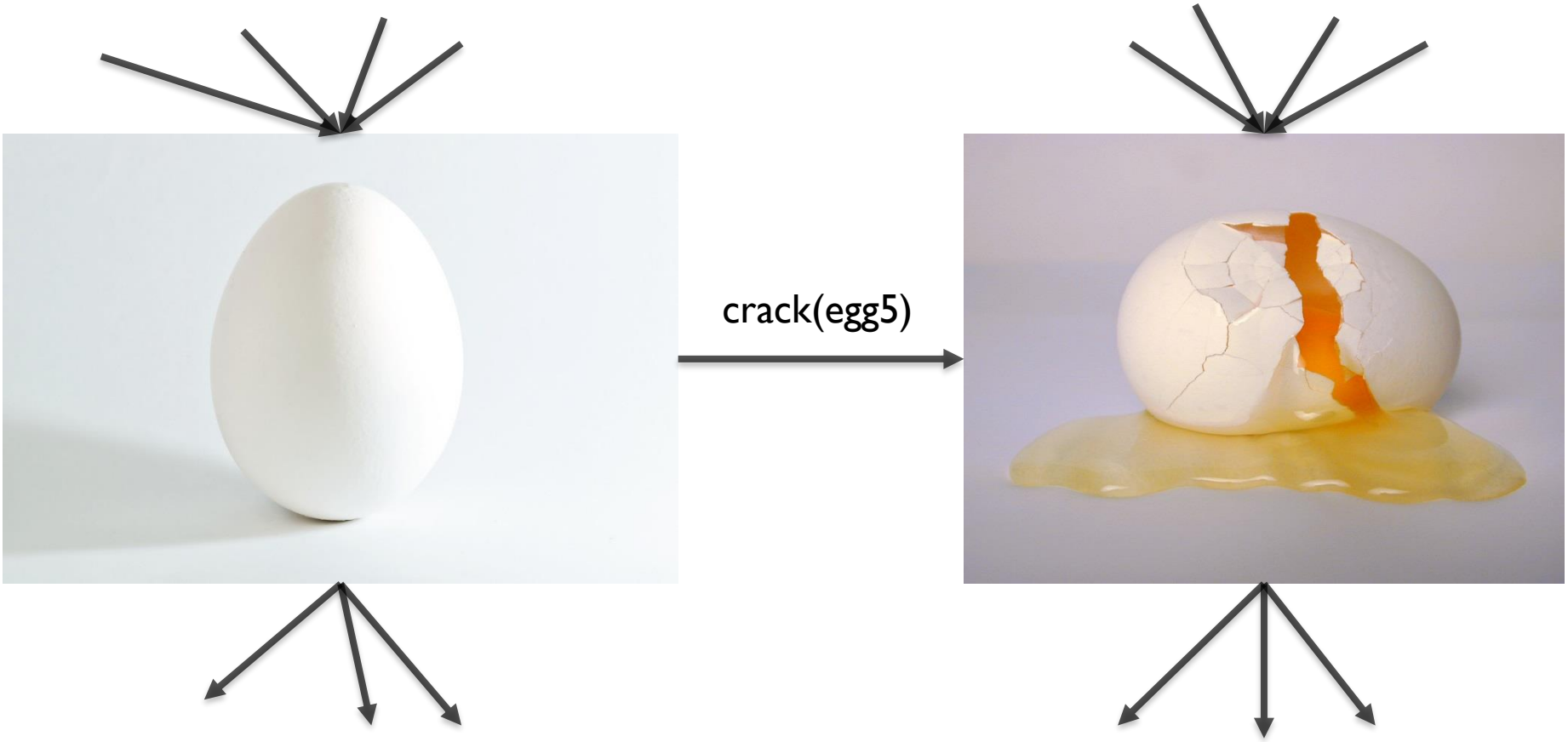
5) Plan extraction: *Generate the sequence of all actions on the path to the solution node*

General Properties of the State Space

State Space: Not Always Symmetric

jonkv@ida

- Example: Unable to return

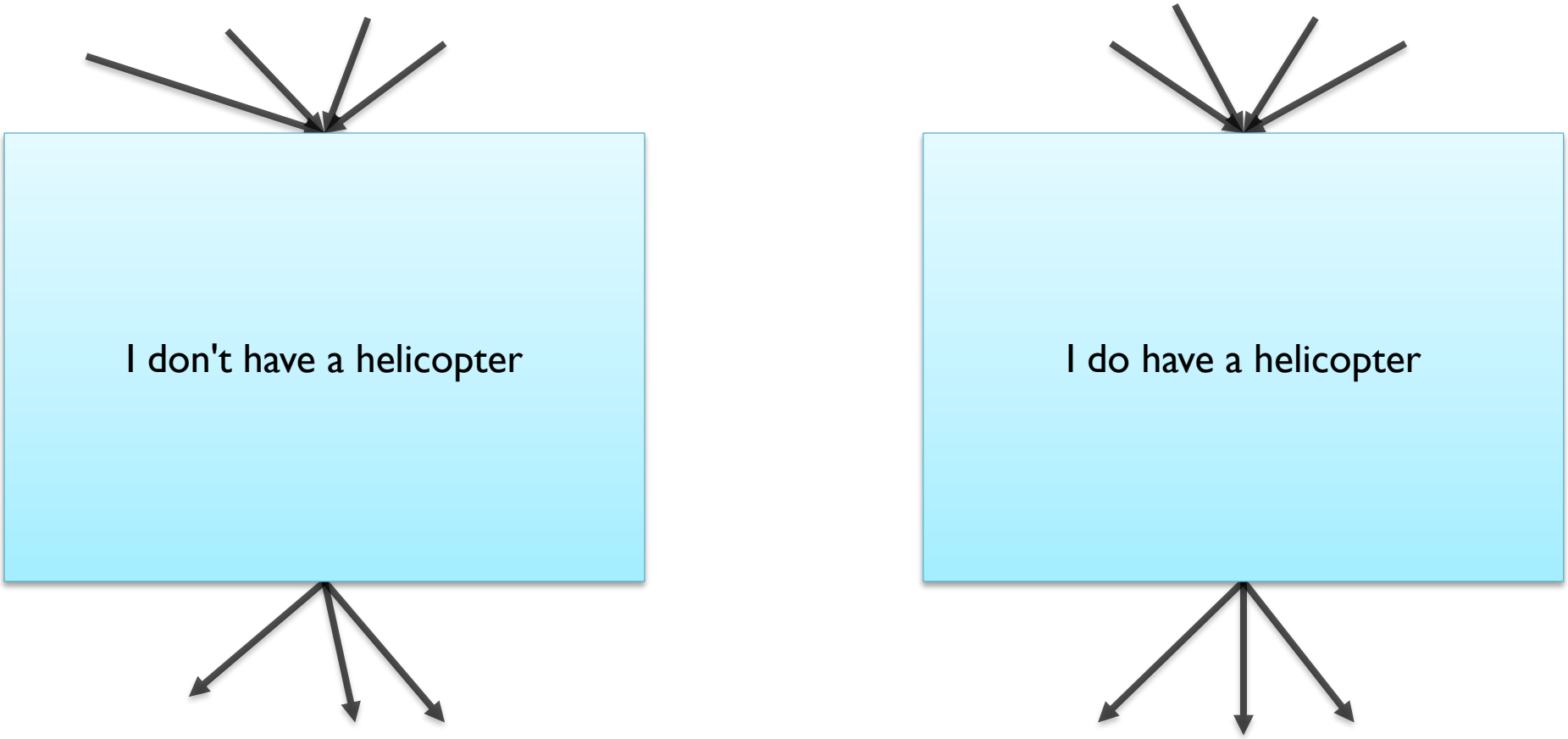


Can never return to the leftmost part of the state space

State Space: Not Always Connected

jonkv@ida

- Example: Disconnected parts of the state space



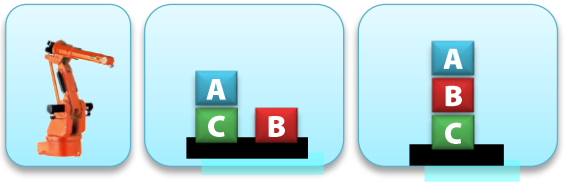
No action for buying a helicopter, no action for losing it
➔ Will stay in the partition where you started!

Exploring the State Space

About Examples

jonkv@ida

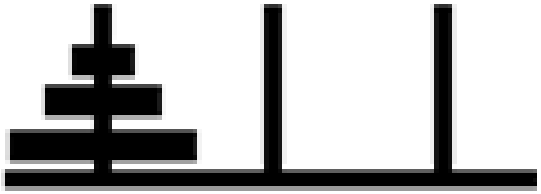
- Exploring the state space… of what?
  - As usual: **toy examples** in **very simple domains**
    - To learn *fundamental principles*
    - To focus on *algorithms and concepts*, not domain details
    - To create *readable, comprehensible examples*
  - Always remember:
    - Real-world problems are larger, more complex



ToH 1: Intuitions

jonkv@ida

- Our **intuitions** often identify states that **we** think are:
  - ”Normal”
  - ”Expected”
  - ”Physically possible”

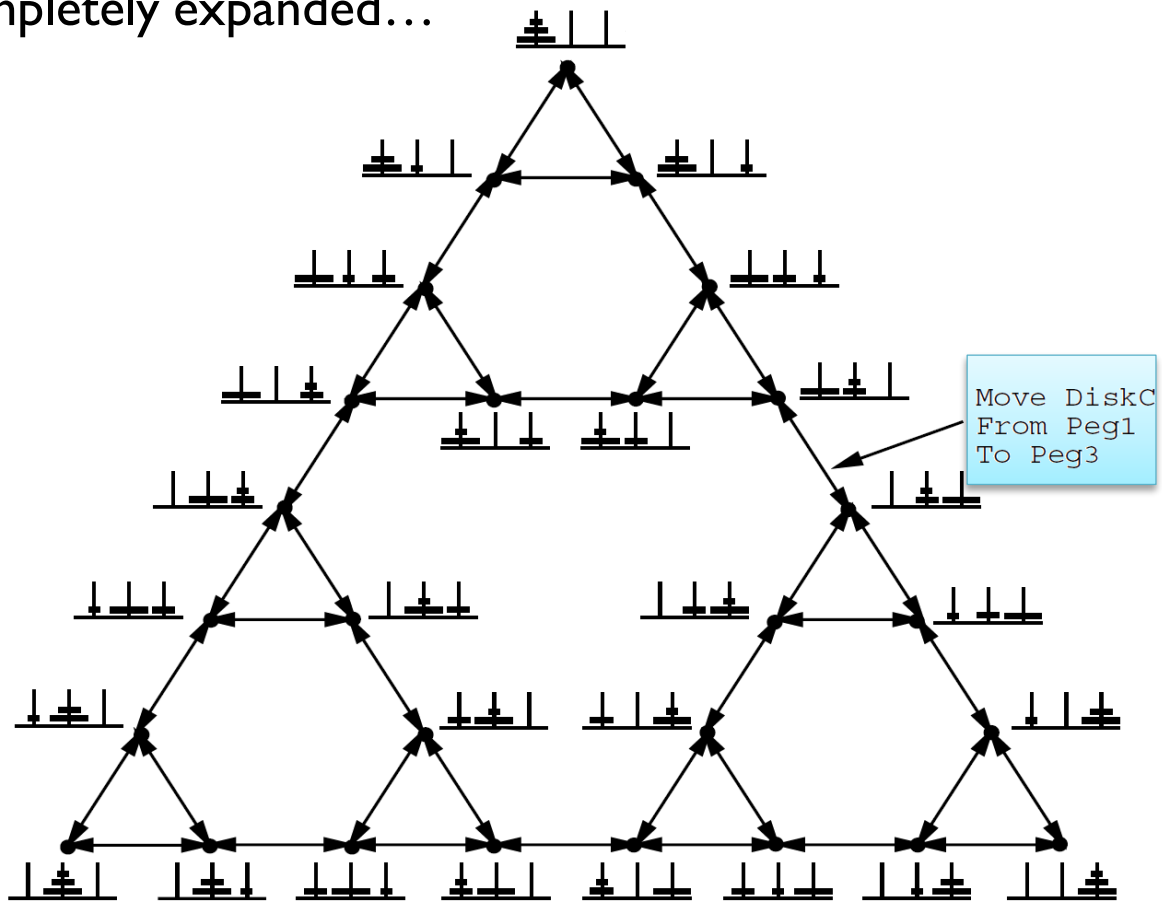


- Usually:
  - The **initial state** is one of those states
  - Mainly need to care about all states **reachable** from there (using the defined actions) – discussed later

ToH 2: What we expect

jonkv@ida

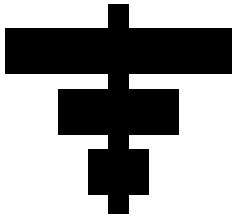
- In ToH (3 pegs, 3 disks), we might **expect** the following 27 states
  - If it is completely expanded…



ToH 3: Against our Intuitions

jonkv@ida

- But given our *definitions*, **every** combination of **facts** is a state
  - Depending on the **formulation**, some ”forbidden” states typically exist
    - *Towers of Hanoi*:



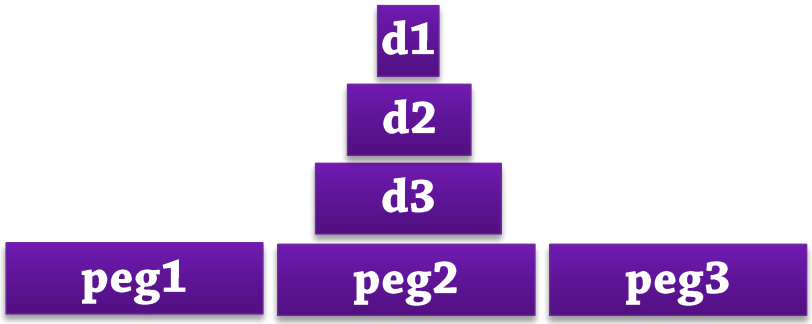
  - The *Blocks World* can have ”counter-intuitive” states where:
    - **holding(A)** and **ontable(A)** are true at the same time

These **ground atoms** are like ”variables” that **can** independently be true or false!

ToH 4: Modeling

jonkv@ida

- ”Depending on the **formulation**” ➔ We need a ToH formulation
  - Let’s begin with a **modeling trick**:



Disks and pegs are "equivalent"
Pegs are the *largest disks*,
so they cannot be moved

ToH 5: Modeling (2)

jonkv@ida

- One version of Towers of Hanoi (PDDL):
  - (**define** (**domain** hanoi)
    (:**requirements** :**strips**)
    (:**predicates** (clear ?x) (on ?x ?y) (smaller ?x ?y))

    (:**action** move
      :**parameters** (?disc ?from ?to)
      :**precondition** (**and** (smaller ?to ?disc) (on ?disc ?from) (clear ?disc) (clear ?to))
      :**effect** (**and** (clear ?from) (on ?disc ?to) (**not** (on ?disc ?from)) (**not** (clear ?to))))
)

clear:	"nothing on top of x"
on:	"x on top of y"
smaller:	"y is smaller than x"

  - (**define** (**problem** hanoi3) (:**domain** hanoi)
    (:**objects** peg1 peg2 peg3 d1 d2 d3)
    (:**init**
      (smaller peg1 d1) (smaller peg1 d2) (smaller peg1 d3)
      (smaller peg2 d1) (smaller peg2 d2) (smaller peg2 d3)
      (smaller peg3 d1) (smaller peg3 d2) (smaller peg3 d3)
      (smaller d2 d1) (smaller d3 d1) (smaller d3 d2)
      (clear peg2) (clear peg3) (clear d1)
      (on d3 peg1) (on d2 d3) (on d1 d2))
    (:**goal** (**and** (on d3 peg3) (on d2 d3) (on d1 d2)))
)

ToH 6: Number of States

jonkv@ida

- How many **states** exist for this problem formulation?
  - (**define** (**domain** hanoi)
    (:**requirements** :**strips**)
    (:**predicates** (clear ?x) (on ?x ?y) (smaller ?x ?y))

    (:**action** move
      :**parameters** (?disc ?from ?to)
      :**precondition** (**and** (smaller ?to ?disc) (on ?disc
      :**effect** (**and** (clear ?from) (on ?disc ?to) (**not** (or
)
  - (**define** (**problem** hanoi3) (:**domain** hanoi)
    (:**objects** peg1 peg2 peg3 d1 d2 d3)
    (:**init**
      (smaller peg1 d1) (smaller peg1 d2) (smaller peg1
      (smaller peg2 d1) (smaller peg2 d2) (smaller peg2
      (smaller peg3 d1) (smaller peg3 d2) (smaller peg3
      (smaller d2 d1) (smaller d3 d1) (smaller d3 d2)
      (clear peg2) (clear peg3) (clear d1)
      (on d3 peg1) (on d2 d3) (on d1 d2))
    (:**goal** (**and** (on d3 peg3) (on d2 d3) (on d1 d2)))
)

Answer:

Every assignment of values
to the ground atoms
is one state

6 objects
2^6 combinations of "clear"
2^{6*6} combinations of "on"
2^{6*6} combinations of "smaller"

2^{78} combinations in total:
302231′454903′657293′676544

The state is just a data structure

Every value combination is a state

ToH 7: Alternatives

jonkv@ida

- Space size for our first formulation:

2^{78} combinations in total:
302231′454903′657293′676544

- Suppose we don't include irrelevant combinations of known, **fixed** predicates ("smaller")?

2^{6} combinations of "clear"
2^{6*6} combinations of "on"

2^{42} combinations in total:
4′398046′511104

- Suppose we **get rid of "clear"** (redundant!)
  - Use more expressive planner
  - (clear ?x) ➔ (not (exists ?y) (on ?y ?x))

2^{6*6} combinations of "on"

2^{36} combinations in total:
68719′476736

- Suppose we **remodel "on"**:
  - below_d1 ∈ {peg1, peg2, peg3, d2, d3}
  - below_d2 ∈ {peg1, peg2, peg3, d1, d3}
  - below_d3 ∈ {peg1, peg2, peg3, d1, d2}

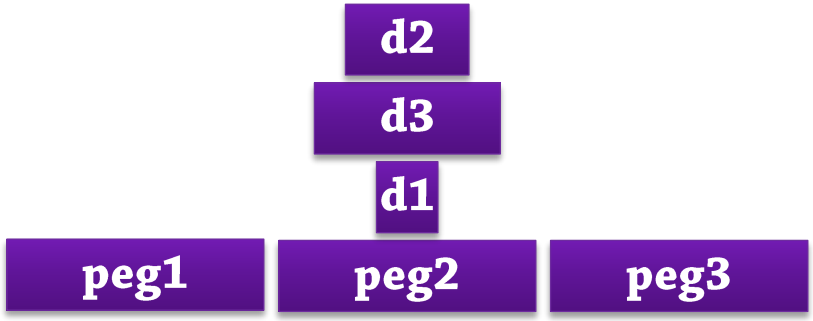
5^{3} combinations in total:
125

Why the extreme dependence on the formulation?

Model Dependence 1

jonkv@ida

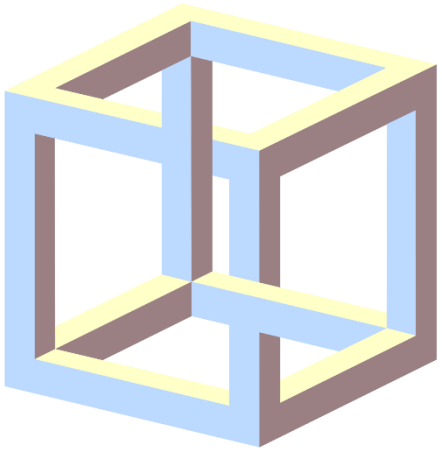
- In all suggested formulations, **this** is one possible state
  - Planners *should not generate* such states,
    but they still *exist*



Model Dependence 2

jonkv@ida

- In some formulations, states such as *this* exist
  - (**and**
    (on peg1 peg2)
    (on d1 d2)
    (on d2 d1)
    (on d3 d3)
    …
)



- In the last formulation example:
  - **below_peg1** does not exist
  - **below_d3** cannot be **d3**
  - (But we can still have circularities)

below_d1 ∈ {peg1, peg2, peg3, d2, d3}
below_d2 ∈ {peg1, peg2, peg3, d1, d3}
below_d3 ∈ {peg1, peg2, peg3, d1, d2}

Some formulations allow more ”unintended states” than others!

Does the size of the state space matter?

Reachability

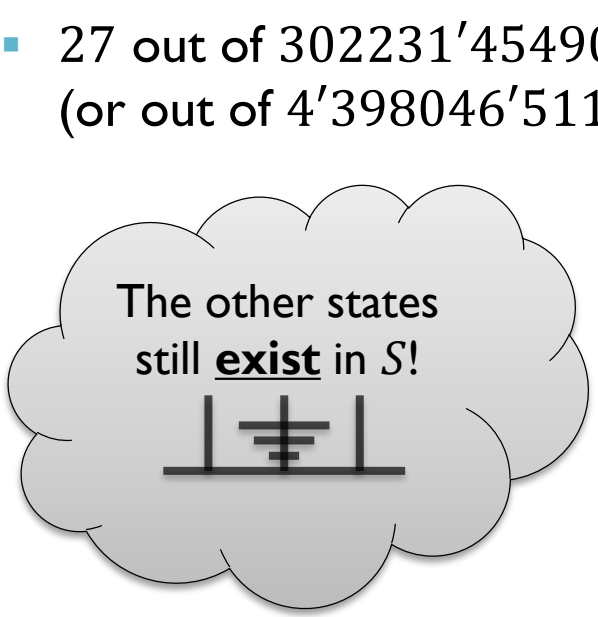
jonkv@ida

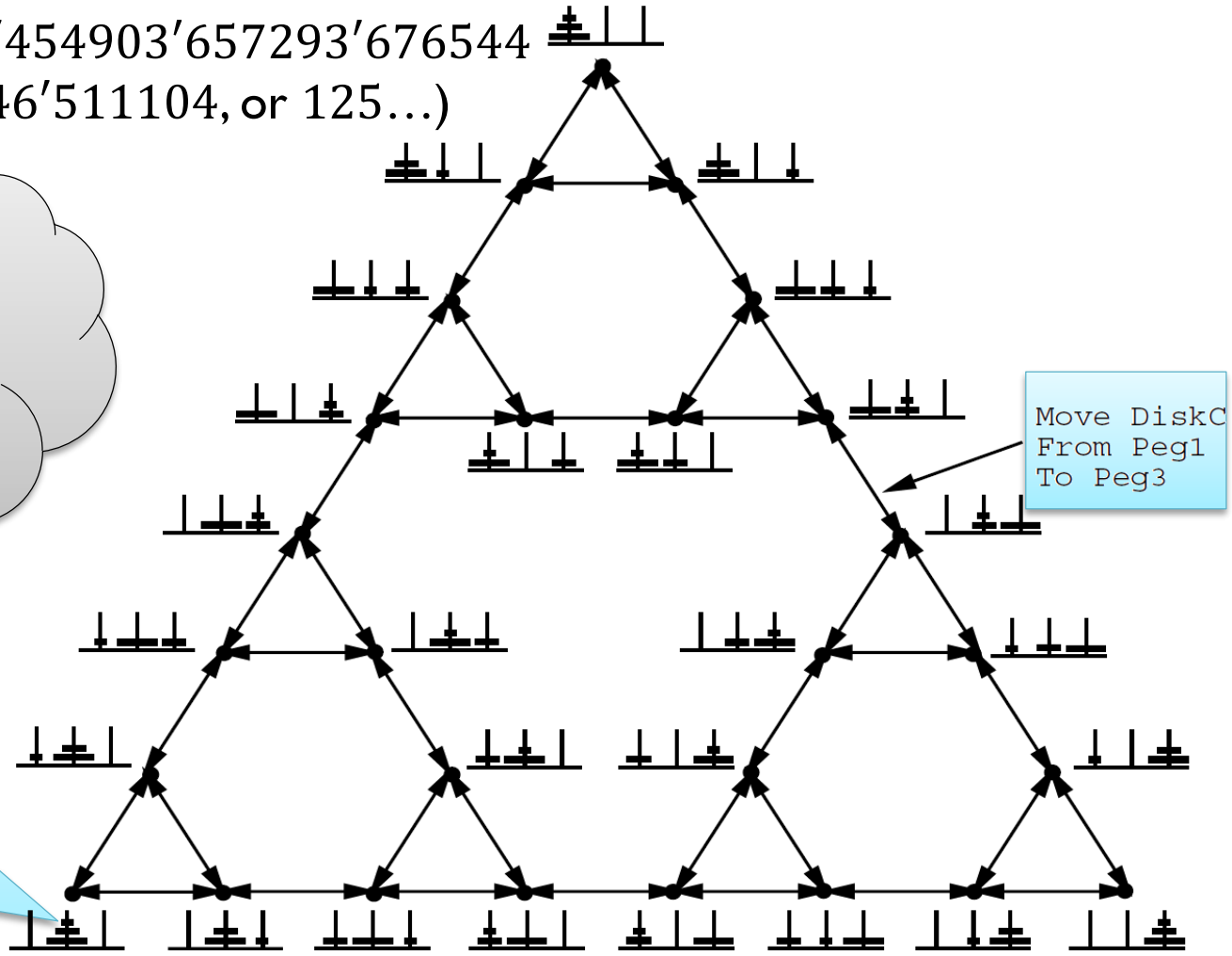
- **Forward state space search:**
  - Will incrementally generate (only) **reachable** states
  - Many unreachable states?
    - More state variables ➔ somewhat more expensive to generate / store a state
  - **Uninformed** strategies (depth first, Dijkstra, …):
    - No difference in what is explored
  - **Informed** forward state space search (A*, hill climbing, …):
    - **Heuristics** might work better with less redundant formulations – or worse…
- **Other search spaces (backward, POCL, temporal, …):**
  - Depends!

Reachability (2): From Initial State

jonkv@ida

- How many **states** are **reachable from** the given **initial state**, using the given actions?
  - 27 out of 302231′454903′657293′676544 (or out of 4′398046′511104, or 125…)





s_{17} clear(peg1) is true
clear(peg2) is false
clear(peg3) is false
on(d1,peg1) is false
on(d3,peg22) is true
…

Reachability (3): From Somewhere!

jonkv@ida

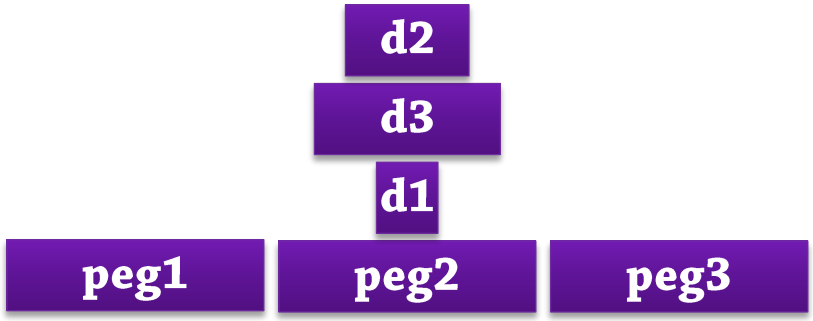
States are not **inherently** "reachable" or "unreachable"

They can be reachable **from** a specific **starting point**!

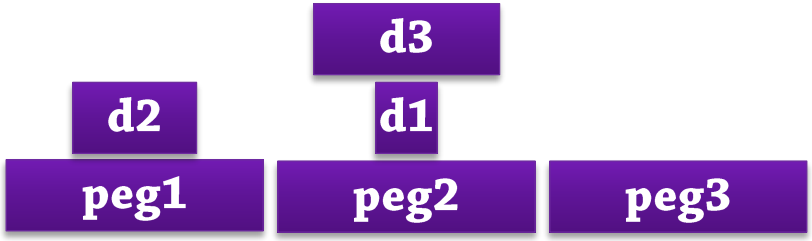
Reachability (4): From 'forbidden' states

jonkv@ida

- Suppose **this** was your initial state
  - *Unreachable* from "all disks in the right order"!



- Then *other* states would be **reachable from this state**
  - If the preconditions hold, then *move* can be applied according to definitions



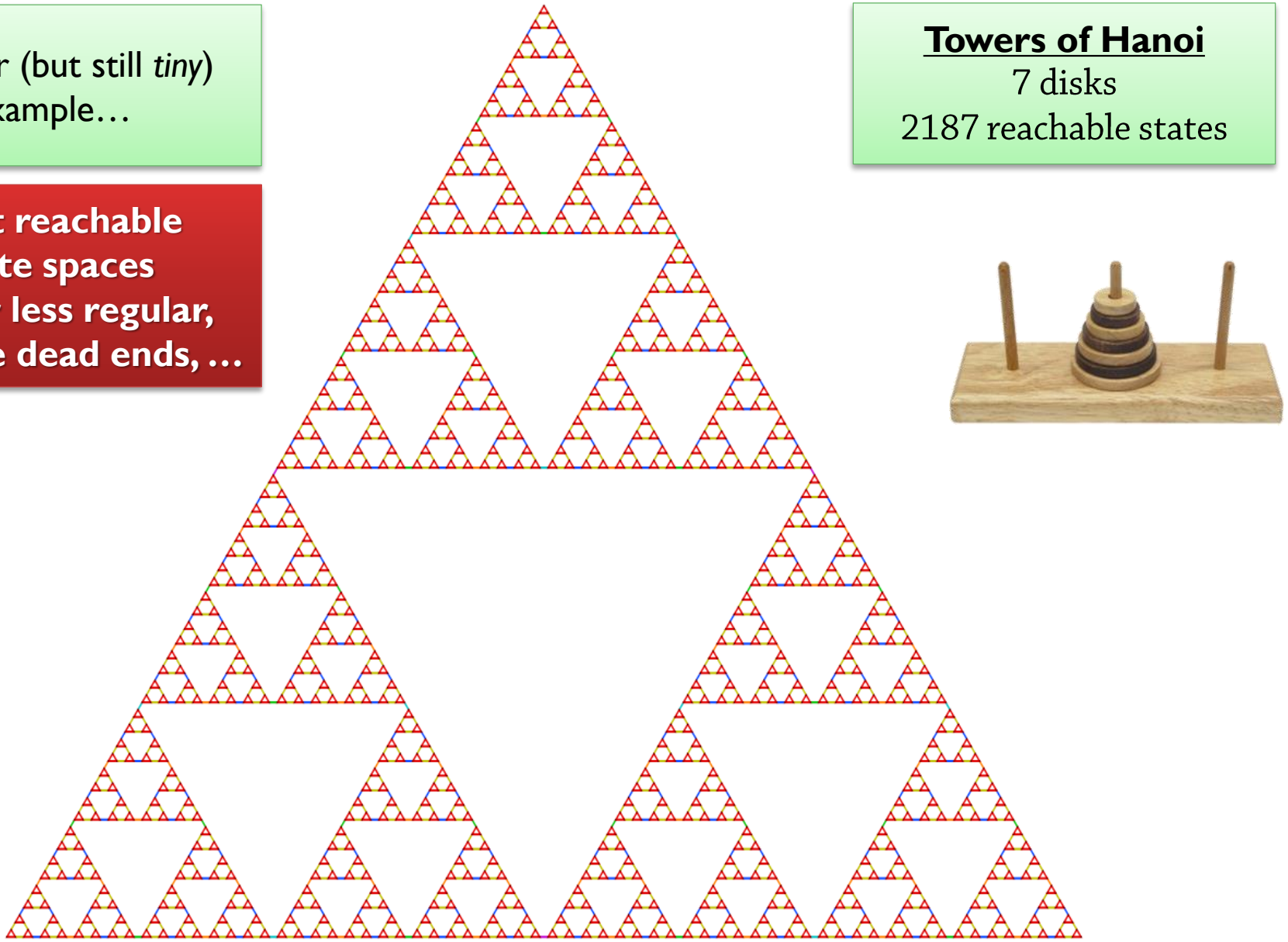
Start in physically realizable state ➔ remain there (assuming correct operators)
Start somewhere else ➔ ???

Reachability (5): Larger

jonkv@ida

A larger (but still *tiny*) example…

Most reachable state spaces are far less regular, can have dead ends, …



Towers of Hanoi
7 disks
2187 reachable states

State Space: Blocks World

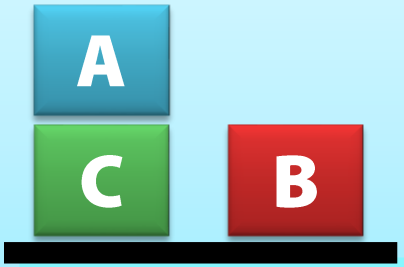
BW 1: Blocks World

jonkv@ida

- Domain 2: The **Blocks World**

You

Initial State



Your greatest desire

BW 2: Model

jonkv@ida

- We will generate classical **sequential plans**
  - One object type: **Blocks**
  - A common blocks world version, with **4 operators**
    - (**pickup** ?x) – takes ?x from the table
    - (**putdown** ?x) – puts ?x on the table
    - (**unstack** ?x ?y) – takes ?x from on top of ?y
    - (**stack** ?x ?y) – puts ?x on top of ?y
  - Predicates used:
    - (**on** ?x ?y) – block ?x is on block ?y
    - (**ontable** ?x) – ?x is on the table
    - (**clear** ?x) – we can place a block on top of ?x
    - (**holding** ?x) – the robot is holding block ?x
    - (**handempty**) – the robot is not holding any block

With n blocks: 2^{n^2+3n+1} states

unstack(A,C) → putdown(A) → pickup(B) → stack(B,C)

BW 3: Operator Reference

jonkv@ida

```
(:action pickup
   :parameters (?x)
   :precondition (and (clear ?x) (on-table ?x)
                      (handempty))
   :effect
   (and (not (on-table ?x))
        (not (clear ?x))
        (not (handempty))
        (holding ?x)))
```

```
(:action unstack
   :parameters (?top ?below)
   :precondition (and (on ?top ?below)
          (clear ?top) (handempty))
   :effect
   (and (holding ?top)
        (clear ?below)
        (not (clear ?top))
        (not (handempty))
        (not (on ?top ?below))))
```

```
(:action putdown
   :parameters (?x)
   :precondition (holding ?x)

   :effect
   (and (on-table ?x)
        (clear ?x)
        (handempty)
        (not (holding ?x))))
```

```
(:action stack
   :parameters (?top ?below)
   :precondition (and (holding ?top)
                      (clear ?below))
   :effect
   (and (not (holding ?top))
        (not (clear ?below))
        (clear ?top)
        (handempty)
        (on ?top ?below)))
```

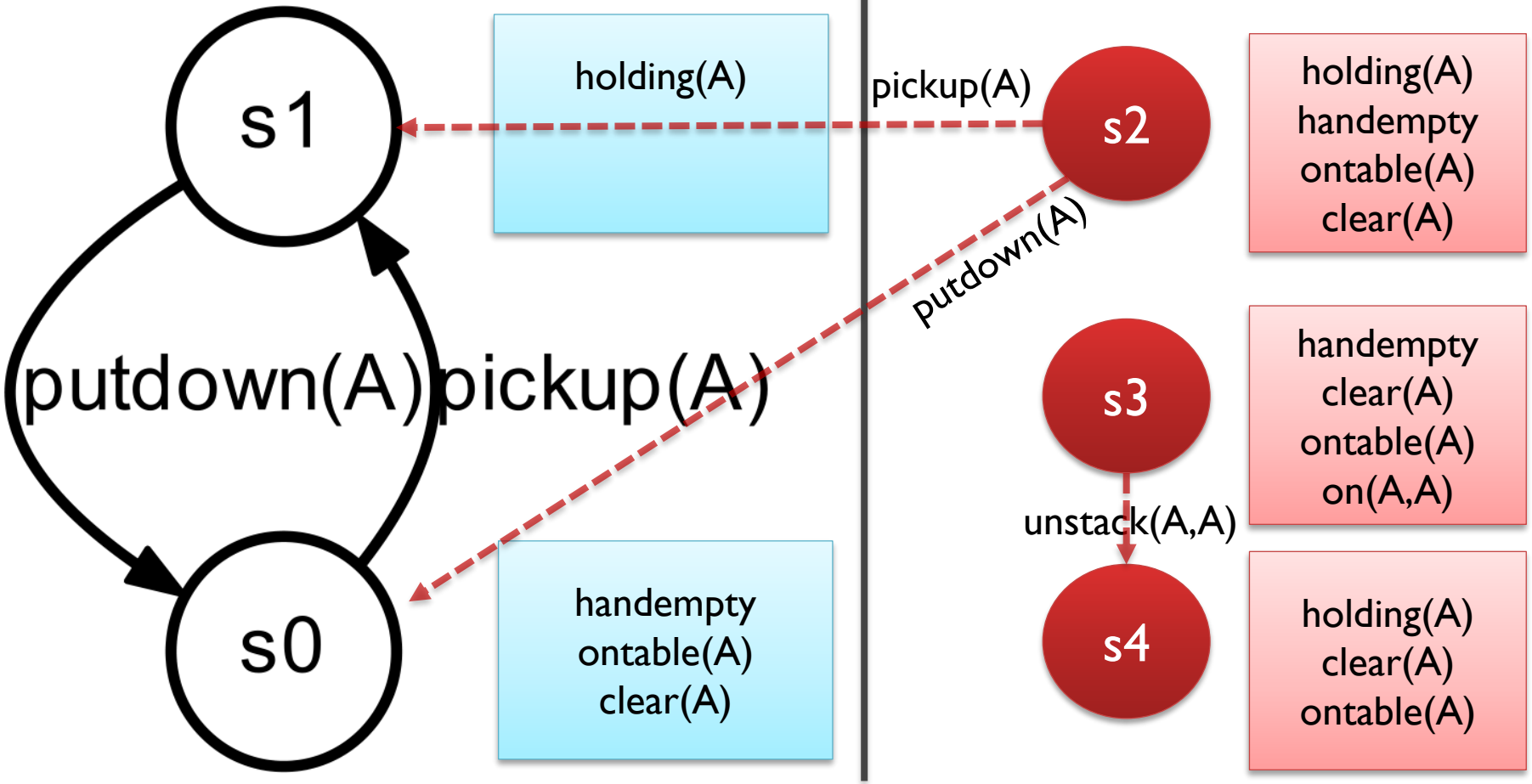
BW 4: Reachable State Space, 1 block

jonkv@ida

We assume we know the initial state
Let's see which states are *reachable* from there!

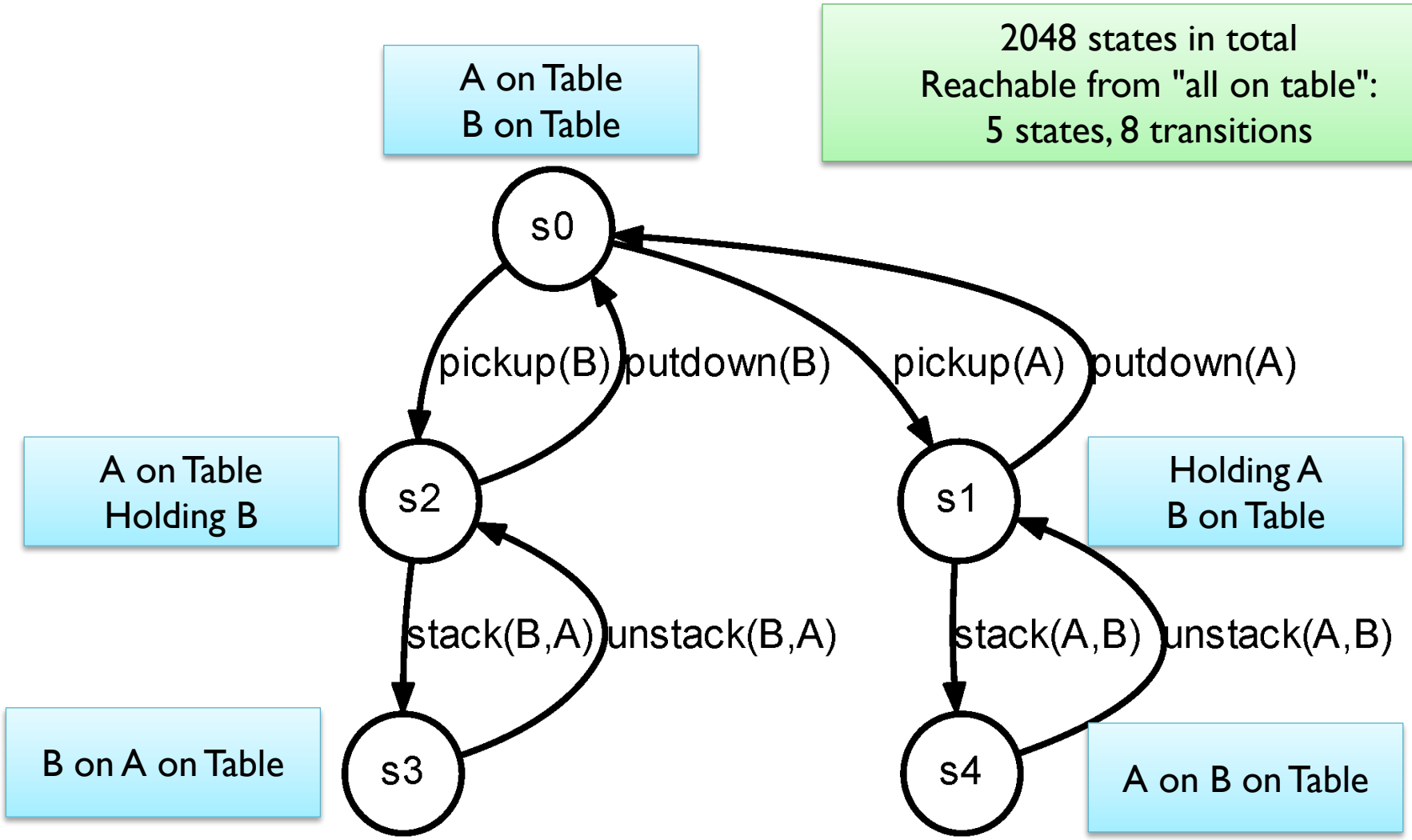
Here: Start with s0 = *all blocks on the table*

Many other states "**exist**",
but are not **reachable**
from the current starting state



BW 5: Reachable State Space, 2 blocks

jonkv@ida

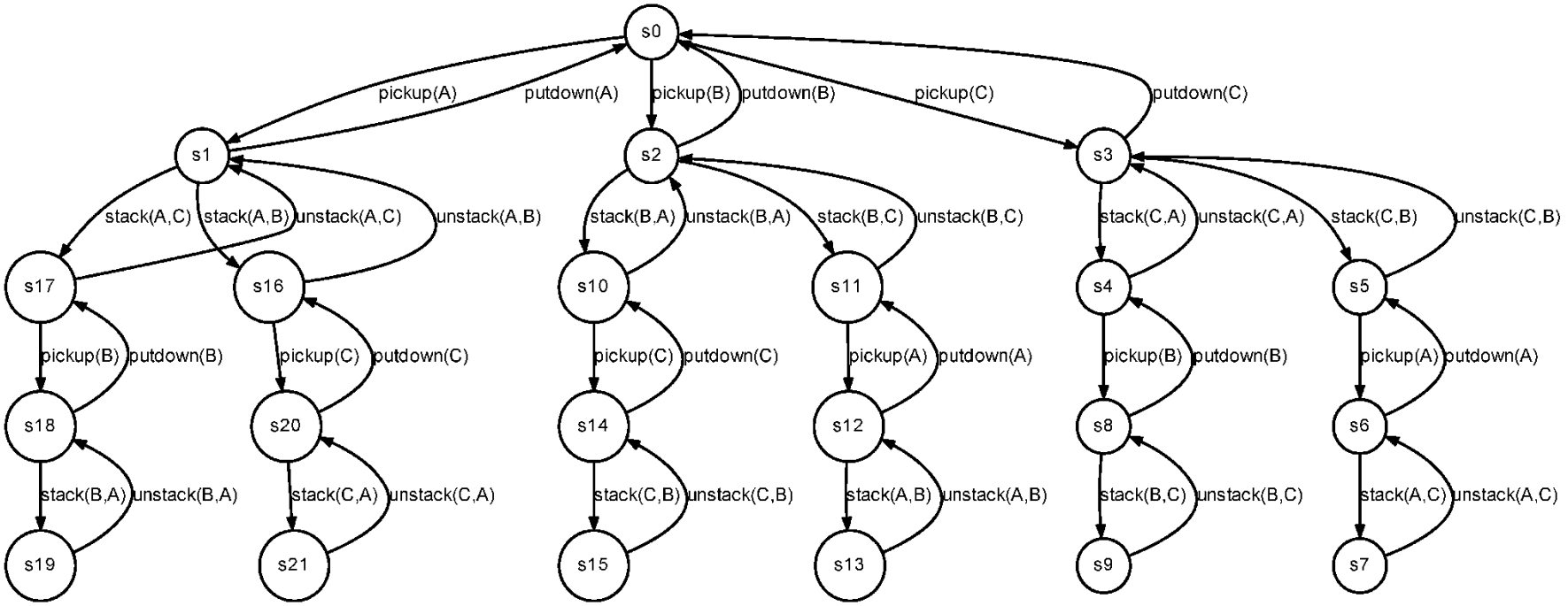


BW 6: Reachable State Space, 3 blocks

jonkv@ida

A on Table
B on Table
C on table

524'288 states in total
Reachable from "all on table":
22 states, 42 transitions

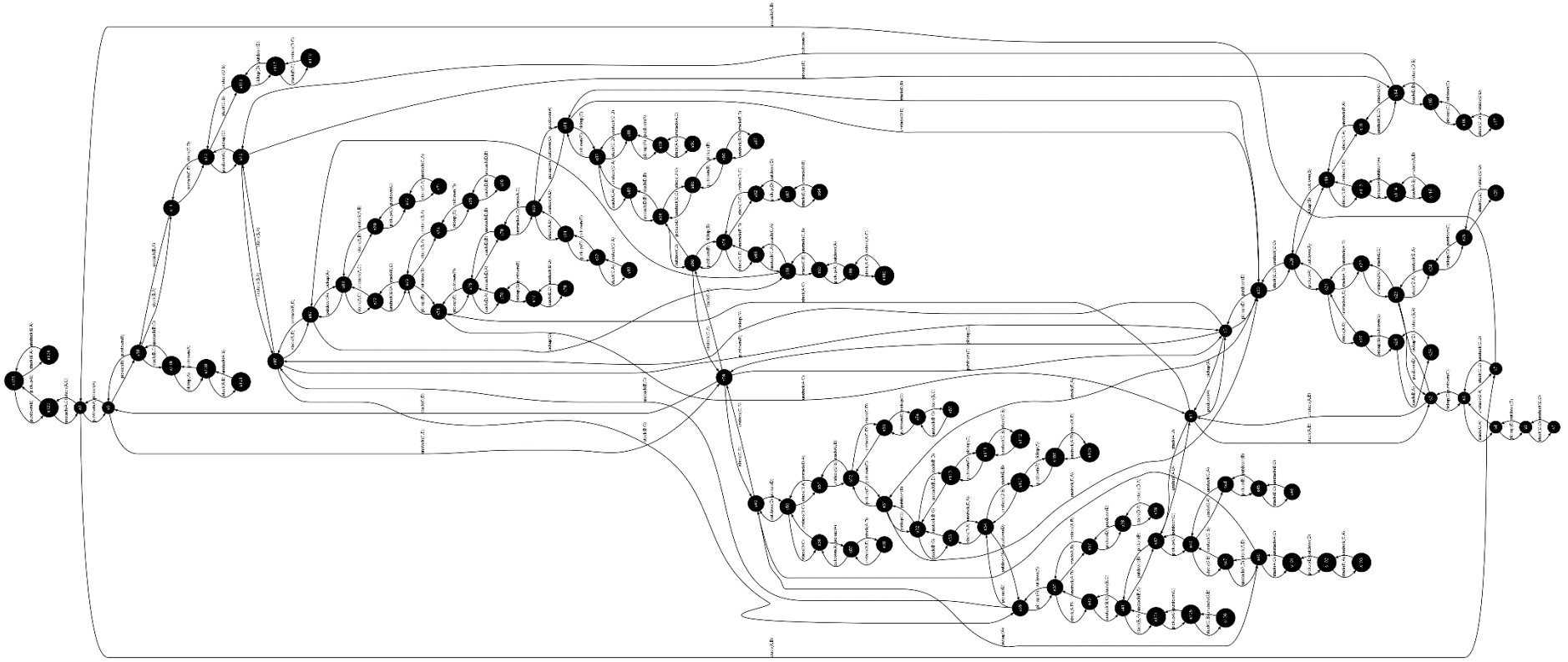


Looking nice and symmetric…

BW 7: Reachable State Space, 4 blocks

jonkv@ida

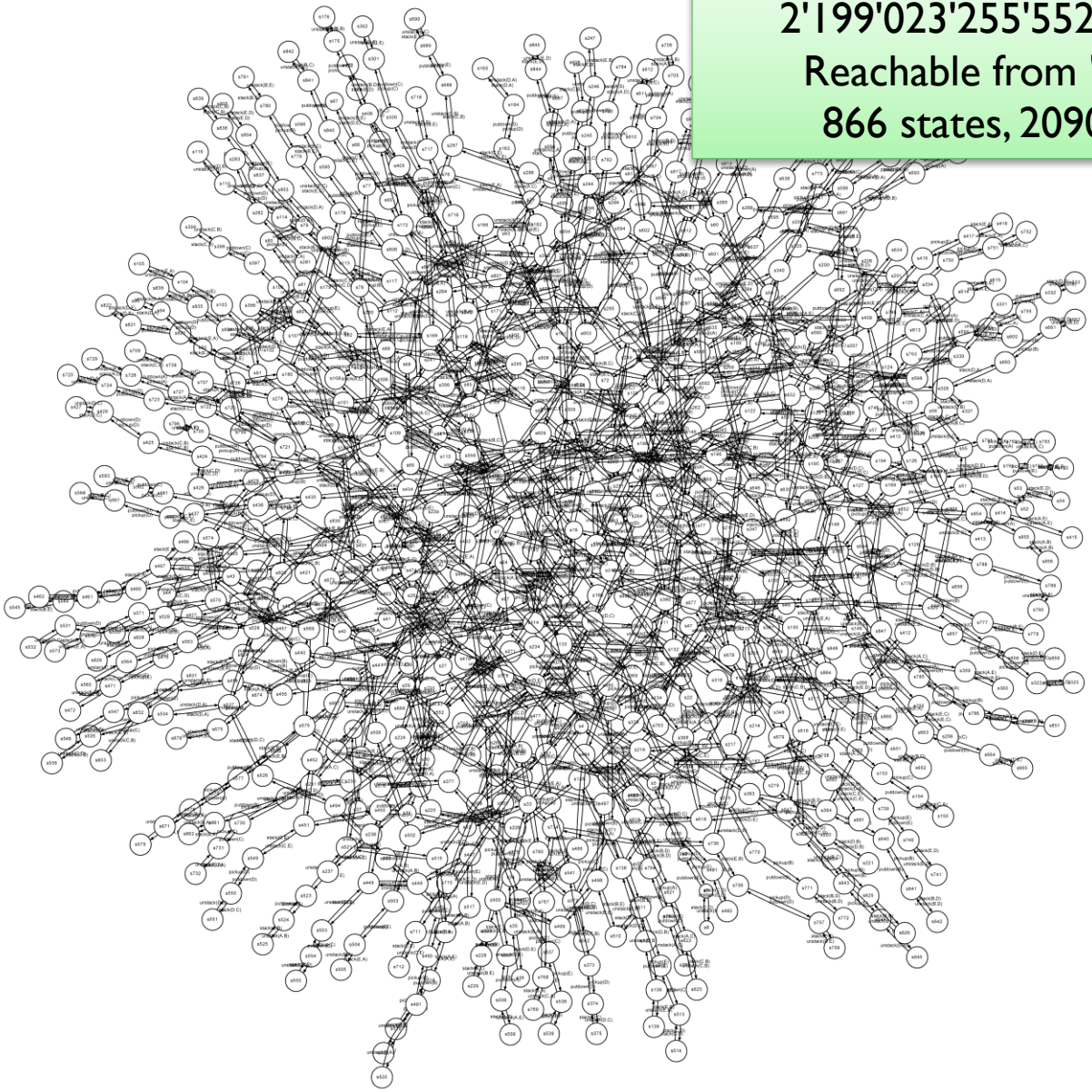
536'870'912 states in total
Reachable from "all on table":
125 states, 272 transitions



BW 8: Reachable State Space, 5 blocks

jonkv@ida

2'199'023'255'552 states in total
Reachable from "all on table":
866 states, 2090 transitions



Blocks World:
Formulations and State Space Sizes

Size 1: Blocks World, PDDL

jonkv@ida

- Standard PDDL predicates:
  - (**on** ?x ?y)
  - (**ontable** ?x)
  - (**clear** ?x)
  - (**holding** ?x)
  - (**handempty**)
- Number of ground atoms, for n blocks:
  - $n^2 + 3n + 1$
- Number of states:
  - 2^{n^2+3n+1}

Size 2: Reachable State Space, sizes 0–10

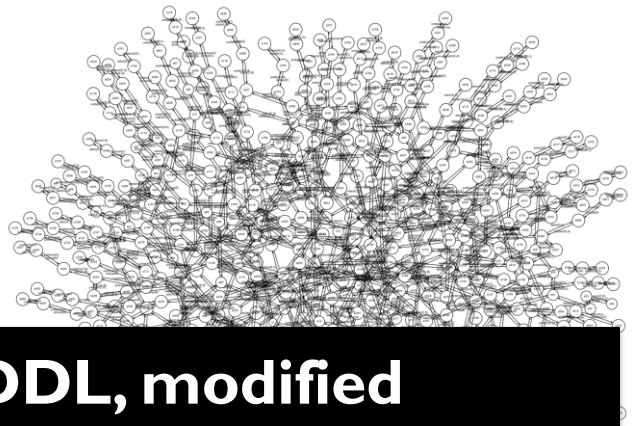
jonkv@ida

Blocks	Ground atoms	States	States reachable from "all on table"	Transitions (edges) in reachable part
0	1	2	1	0
1	5	32	2	2
2	11	2048	5	8
3	19	524288	22	42
4	29	536870912	125	272
5	41	2199023255552	866	2090
6	55	360287970189639 68	7057	18552
7	71	2361183241434822606848	65990	186578
8	89	6189700196426901374495621 12	695417	2094752
9	109	6490371073168534535663120411525 12	8145730	25951122
10	131	2722258935367507707706996859454145691648	…	…

Size 3: : Formulations (1)

jonkv@ida

- Example: Blocks world with 5 blocks

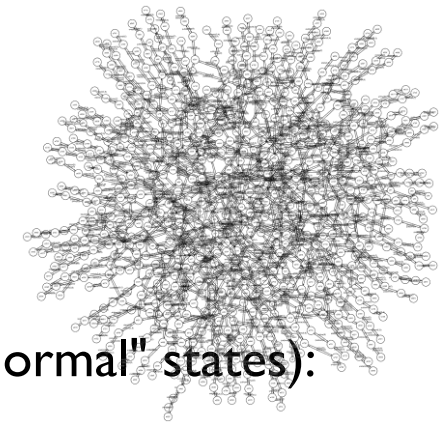


Standard PDDL	PDDL, modified
	Omit (ontable ?x), (clear ?x) In *physically achievable* states, can be *deduced* from (on ?x ?y), (holding ?x)
2^{n^2+3n+1}	2^{n^2+n+1}
2'199'023'255'552 states (reachable and unreachable)	**2'147'483'648 states** (reachable and unreachable)
866 reachable	**866** reachable

BW: Formulations (2)

jonkv@ida

- Example: Blocks world with 5 blocks
  - **2'199'023'255'552** or **2'147'483'648 states** in the standard predicate representation



  - But in **all 866 states reachable** from "all-on-table" (all "normal" states):
    - Any state satisfies **exactly one** of the following – a clique:
      - **(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
      - **(on E A)** – Below E

Provides more structure: Obvious that A can't be under B and under C

Useful in some situations, such as PDB heuristics

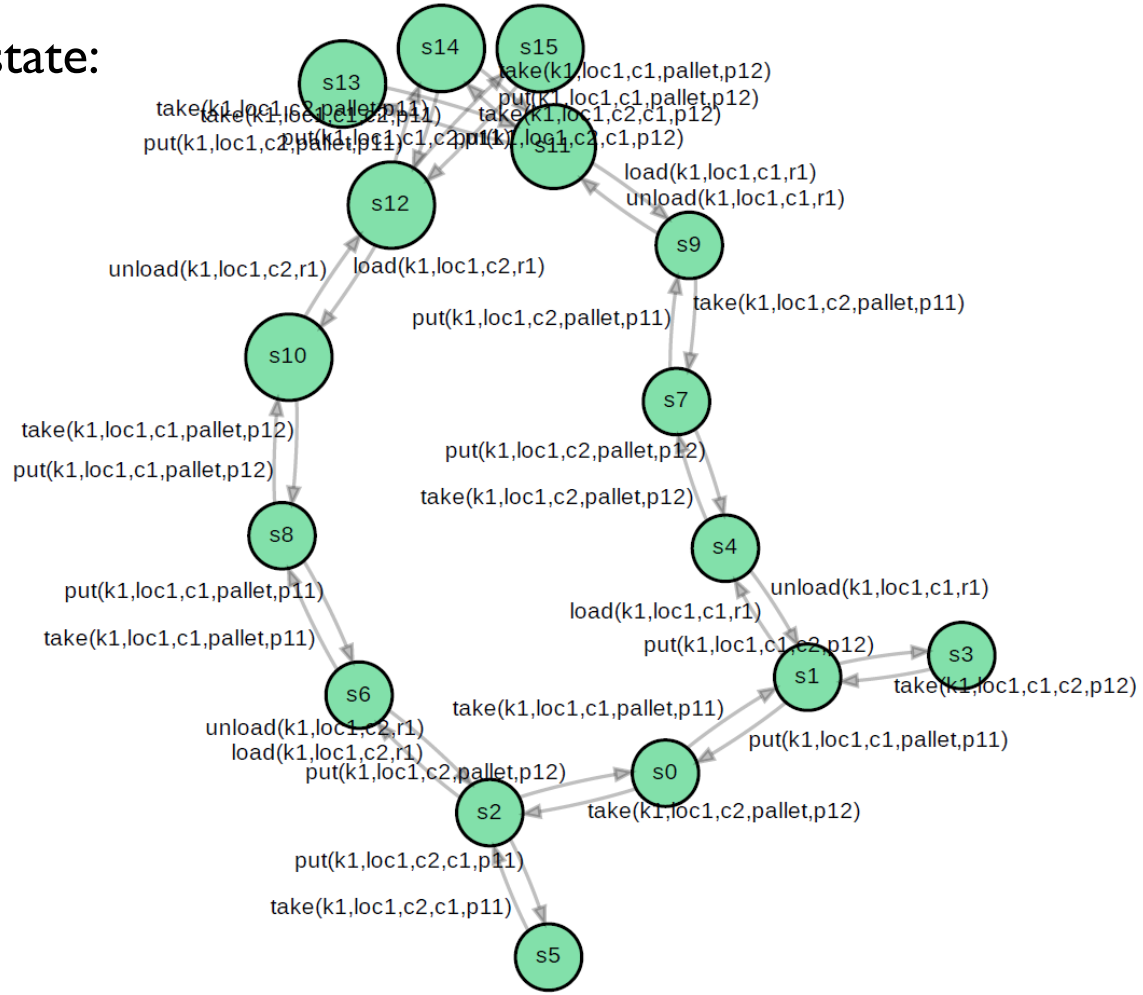
    - Remove those facts, introduce state variables (same for other blocks):
      - **aboveA ∈ { gripper, nothing, B, C, D, E }**
    - Result: $(n+1)^n \cdot 2^{n+1}$ = **497'664** states, 866 reachable

Dock Worker Robots

Dock Worker Robots: Example 1

jonkv@ida

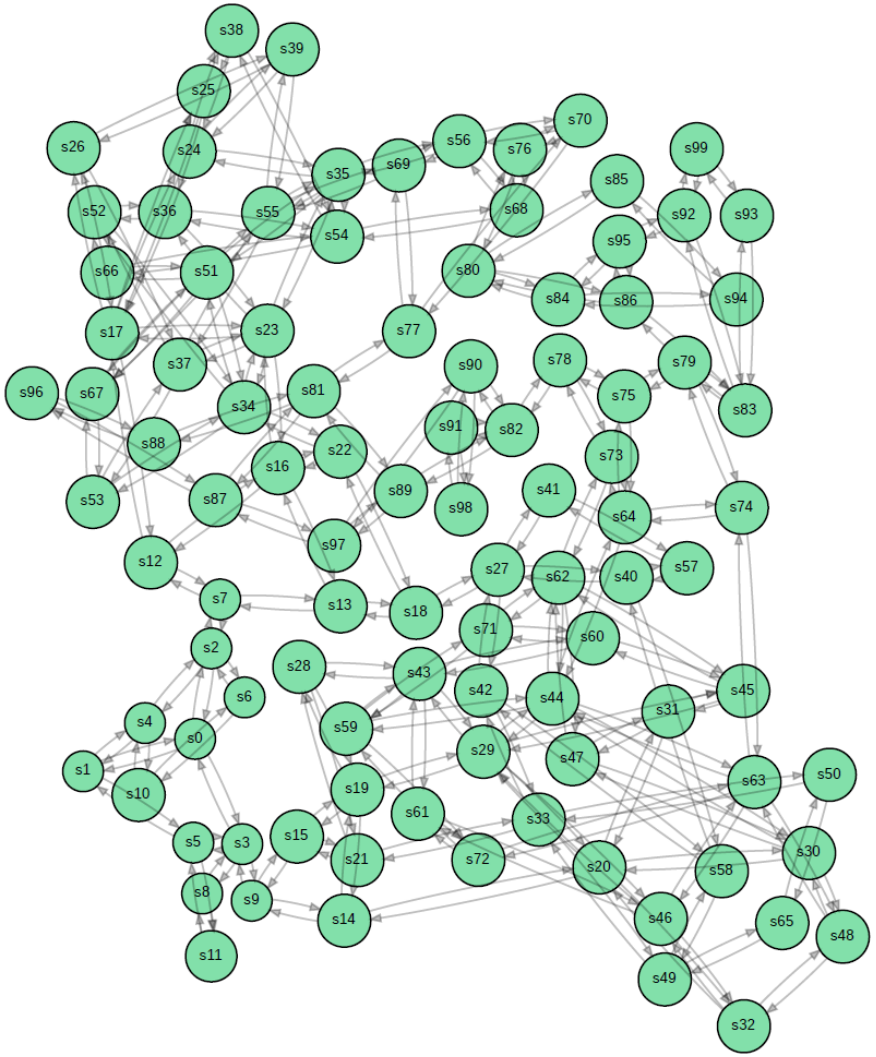
- Example 1: 1 location, 2 piles, 1 robot, 1 crane, 2 containers
  - 2^{35} states
  - Given a particular initial state:
    - 16 states reachable
    - 32 edges reachable



Dock Worker Robots: Example 2

jonkv@ida

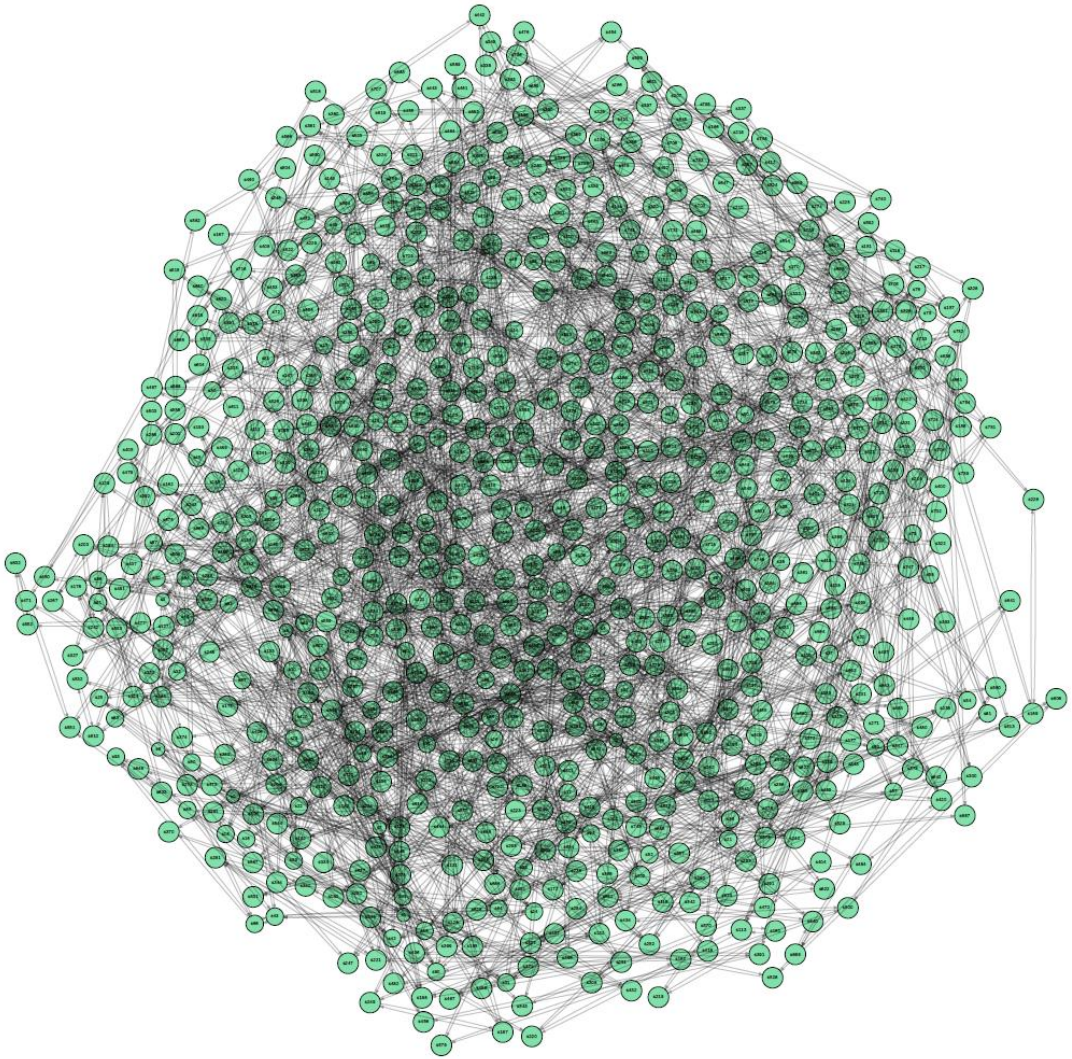
- Example 2: **2 locations, 4 piles,** 1 robot, **2 cranes**, 2 containers
  - 2^{65} states
  - Given a particular initial state:
    - 100 states reachable
    - 332 edges reachable



Dock Worker Robots: Example 3

jonkv@ida

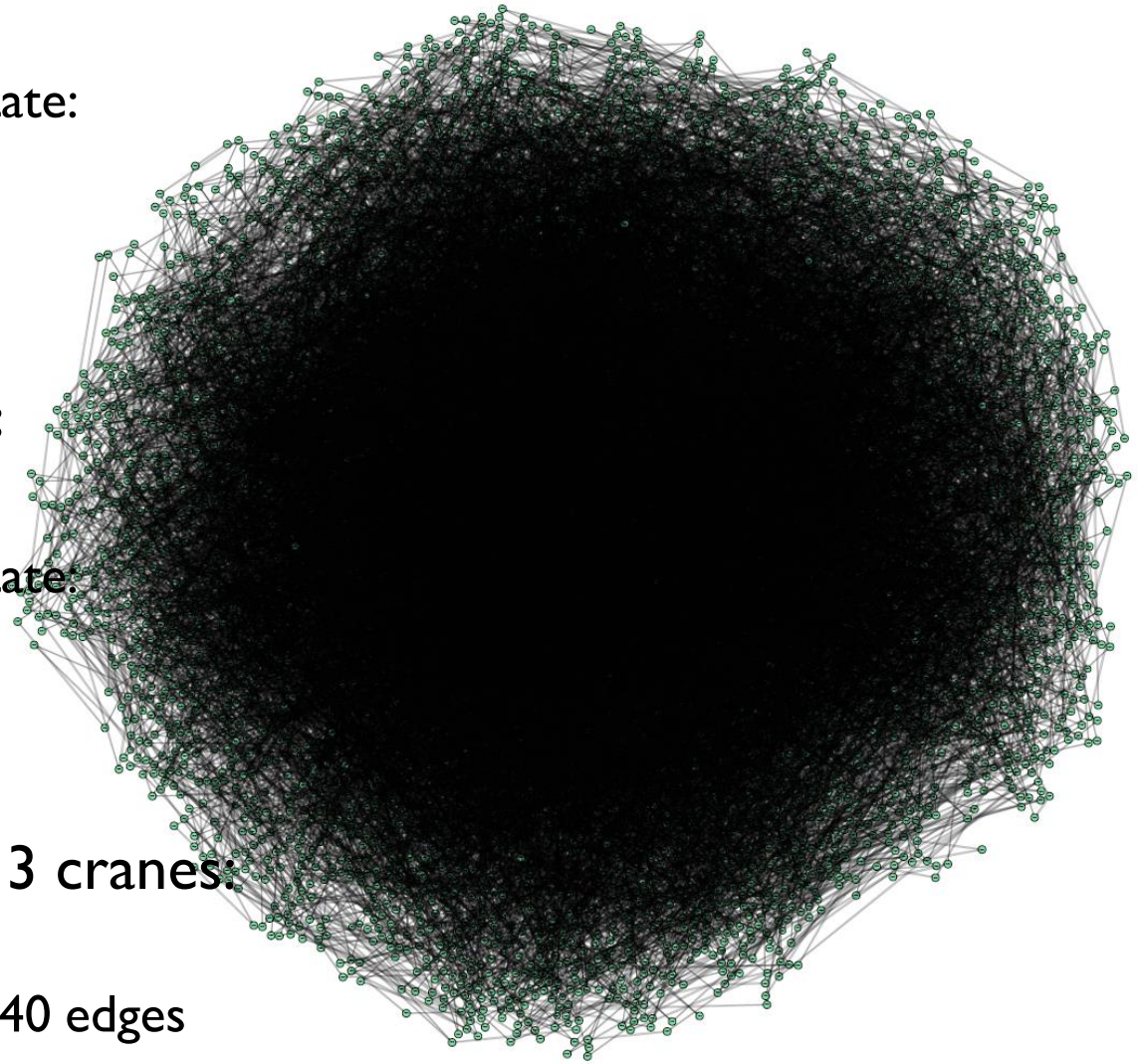
- Example 3: 2 locations, 4 piles, 1 robot, 2 cranes, **3 containers**
  - 2^{83} states
  - Given a particular initial state
    - 756 states reachable
    - 2916 edges reachable



Dock Worker Robots: Example 4

jonkv@ida

- Example 4: 2 locations, 4 piles, 1 robot, 2 cranes, 4 **containers**
  - 2^{103} states
  - Given a particular initial state:
    - 6192 states reachable
    - 25968 edges reachable



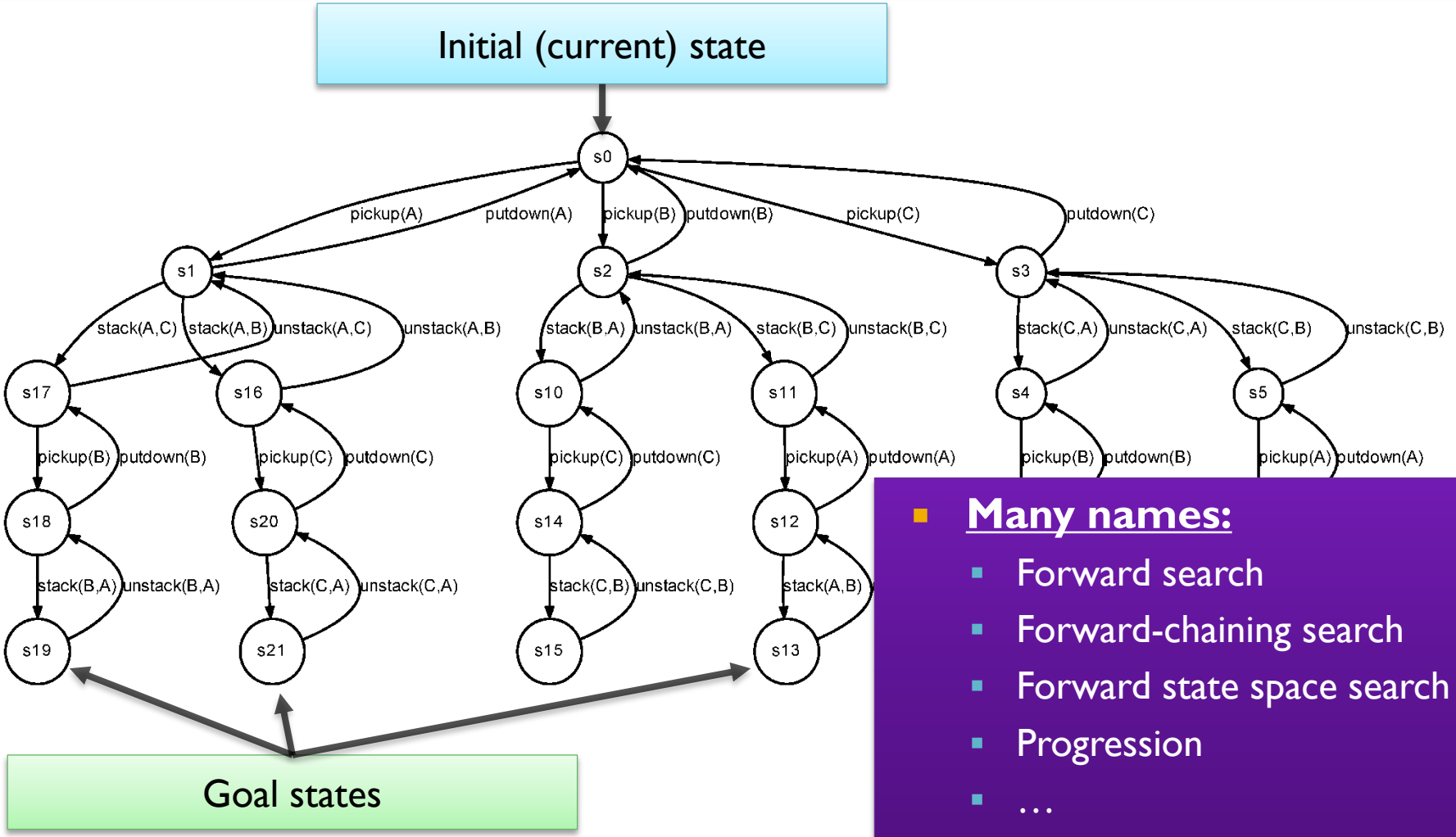
- 6 containers (no image):
  - 2^{149} states
  - Given a particular initial state:
    - 542880 states reachable
    - 2486880 edges reachable
- Also 3 locations, 6 piles, 3 cranes:
  - 2^{207} states,
    1313280 reachable, 6373440 edges

Forward State Space Search

Forward State Space Search

jonkv@ida

Find a path in the forward state space
from the initial state (node) to any goal state

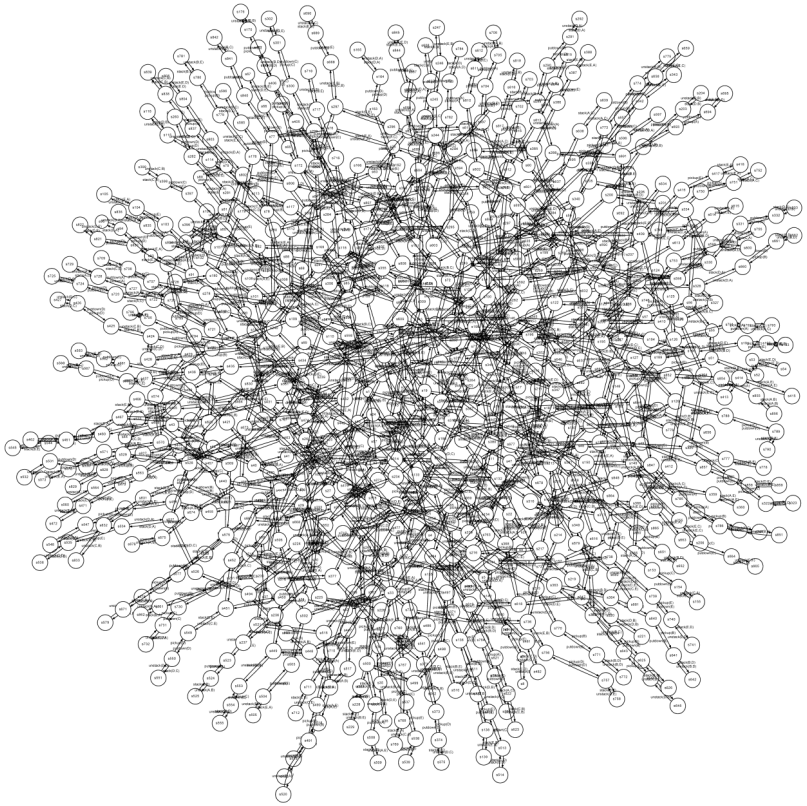


- **Many names:**
  - Forward search
  - Forward-chaining search
  - Forward state space search
  - Progression
  - …

FSSS 2: Don't Precompute

jonkv@ida

- The planner is **not** given a complete precomputed search graph!

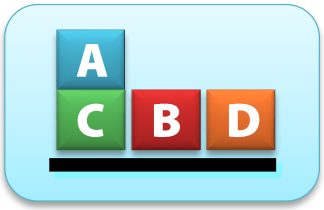


Usually too large!
➔ Generate as we go,
hope we don't actually need the *entire* graph

FSSS 3: Initial state

jonkv@ida

- The **user** (robot?) **observes** the current state of the world
  - The *initial* state



- Must **describe** this using the specified **formal state syntax**…
  - s_0 = { **clear**(A), **on**(A,C), **ontable**(C),
    clear(B), **ontable**(B), **clear**(D), **ontable**(D), **handempty** }

- …and give it to the **planner**, which **[2]** creates **one** search node

{ clear(A), on(A,C), ontable(C),
clear(B), ontable(B), clear(D), ontable(D), handempty }

FSSS 4: Successors

jonkv@ida

- Given **any open search node** (to be selected by a *strategy*)…

  { clear(A), on(A,C), ontable(C),
  clear(B), ontable(B), clear(D), ontable(D), handempty }

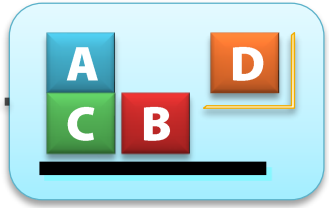
- …we can [3] find **successors** – by **applying applicable actions**!
  - **action** pickup(D)
    - Precondition: **ontable**(D) ∧ **clear**(D) ∧ **handempty** // precond satisfied!
      Effects: ¬**ontable**(D) ∧ ¬**clear**(D) ∧ ¬**handempty** ∧ **holding**(D)

- This generates **new reachable states**…

…which can also be illustrated

{ clear(A), on(A,C), ontable(C),
clear(B), ontable(B), clear(D), ontable(D), handempty }

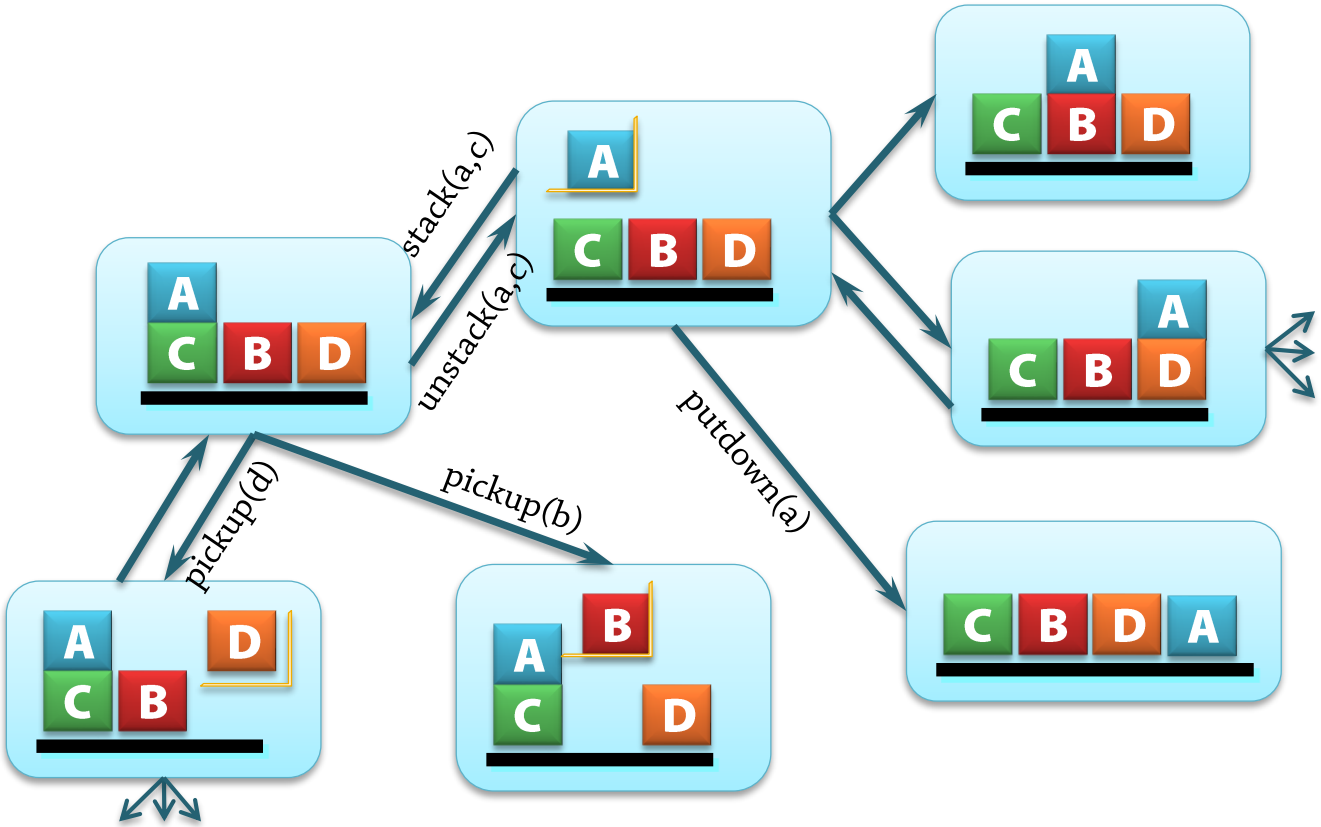
{ clear(A), on(A,C), ontable(C),
clear(B), ontable(B), holding(D) }



FSSS 5: Step by step

jonkv@ida

- A **search strategy** will [6] **choose** which node to expand…
  - [4] Solution criterion: State satisfies goal formula
  - [5] Plan extraction: Extract actions from the path between init and goal state



This is *illustrated* – the planner works with *sets of facts*

Repetition: Planning as Search

jonkv@ida

- **search**(*problem*) {
  initial-node ← **make-initial-node**(*problem*) **// [2]**
  open ← { *initial-node* }
  while (*open* ≠ ∅) {
    node ← **search-strategy-remove-from**(*open*) **// [6]**
    if is-solution(*node*) **then** **// [4]**
      return extract-plan-from(*node*) **// [5]**

  foreach *newnode* ∈ **successors**(*node*) { **// [3]**
    add *newnode* to *open*

  }
  }
  // Expanded the entire search space without finding a solution
  return failure;
}

Expand node

FSSS 6: Instantiated Algorithm

jonkv@ida

- **forward-search**(A, s_0, g) {
  initial-node ← $\langle s_0, \epsilon \rangle$ // [2]
  open ← { *initial-node* }
  while (*open* ≠ ∅) {
    node=$\langle s, \pi \rangle$ ← **search-strategy-remove-from**(*open*) // [6]
    if **is-solution**(*node*) **then** // [4] **check goal formula in state** s
      return π // [5]

  Expand node

    foreach $a \in A$ such that $\gamma(s, a) \neq \emptyset$ { // [3]
      $\{s'\} \leftarrow \gamma(s, a)$
      $\pi' \leftarrow$ **append**(π, a)
      add $\langle s', \pi' \rangle$ to *open*
    }
  }
  // **Expanded the entire search s**
  return failure;
}

Forward search:
Reach in one *step* =
reach by one *action application*

To simplify extracting a plan, nodes above include the plan to reach a state!

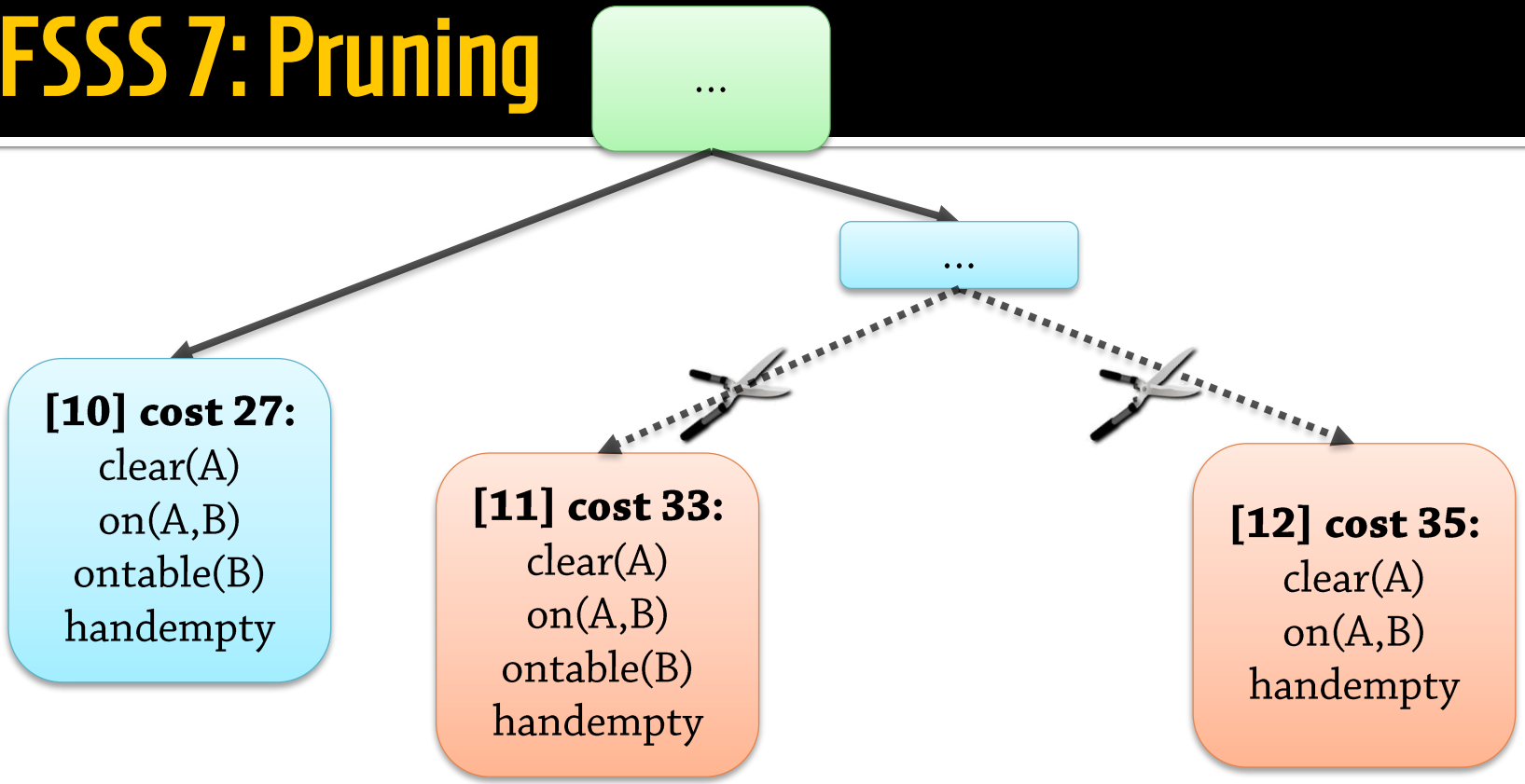
Technically, this searches the space of <state,path> pairs

Still generally called **state space** search…

Is always **sound**
Completeness depends on the strategy

FSSS 7: Pruning

jonkv@ida



Reach a more *expensive* node with the *same* state
➔ can prune
(discard the node without expanding)

If preconditions and goals are **positive**:

Reach a node with a *subset* of the facts
➔ can prune

Allow negative preconds such as (not (ontable B)) ➔
action may be applicable in subtree under [12] but *not* under [11] ➔
must investigate this subtree as well!