Course "Automated Planning: Theory and Practice"
Chapter 04: The Forward State Space

Teacher: Marco Roveri - marco.roveri@unitn.it
M.S. Course: Artificial Intelligence Systems (LM)
A.A.: 2025-2026
Where: DISI, University of Trento
URL: https://shorturl.at/A81hf

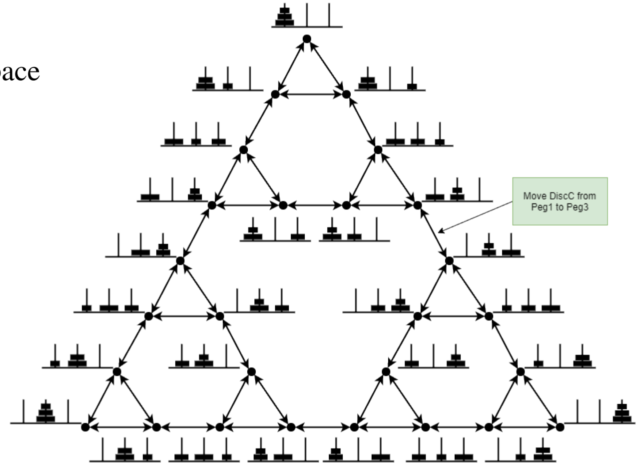
Last updated: Sunday 28th September, 2025

Terms of Use and Copyright

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

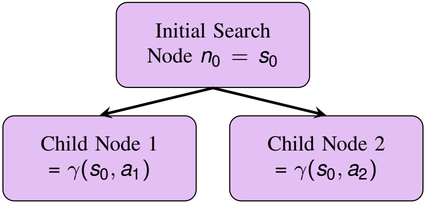
State Space

- 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)

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 node n to a *goal state*, I will be done!

[2] Initial search node:

Corresponds directly to the initial state

[3] Branching rule:

"Forward", "Progression": applying actions in their natural direction!

$\Longrightarrow$ For every action a applicable in state s, generate the state $\gamma(s, a)$

[4] Solution criterion:

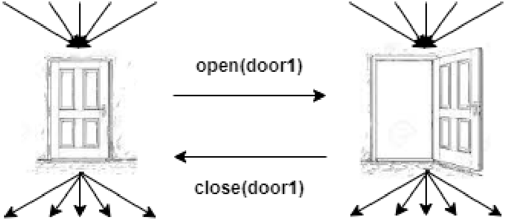
The state of the node *satisfies* the goal formula!

[5] Plan extraction:

Generate the *sequence* of all actions on the path from the initial node to the solution node!

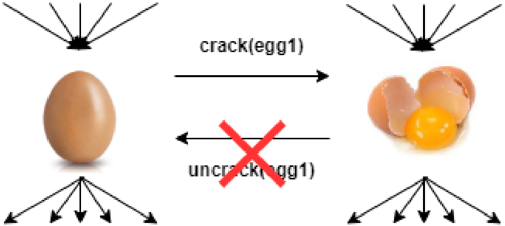
State Space: Not Always Symmetric

- Example: able to return



- Example: unable to return

Can never return to the leftmost part of the search space!



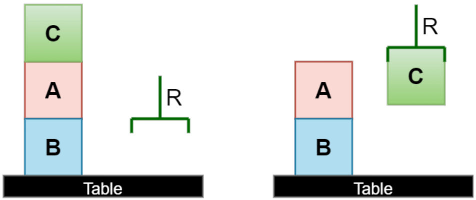
State Space: Not Always Connected

- Example: Disconnected parts of the state space!

No action for buying a rocket, not action for losing it
$\Longrightarrow$ Will stay in the partition where the search ended up!

About Examples

- Exploring the states 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!



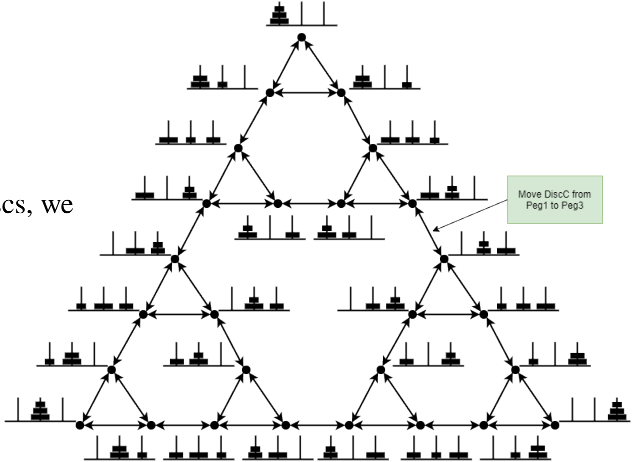
Tower of Hanoi: Intuition

- 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) – we will discuss in detail later!

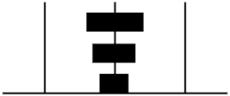
Tower of Hanoi: What we expect

- In Tower of Hanoi with 3 pegs and 3 discs, we may expect 27 states
  - If it is completely expanded...

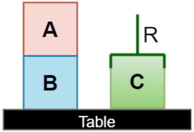


Tower of Hanoi: Against our Intuitions

- But given our *definitions*, every combination of facts is a states
  - Depending on the formulation some "forbidden" states typically exists
    - Tower of Hanoi:



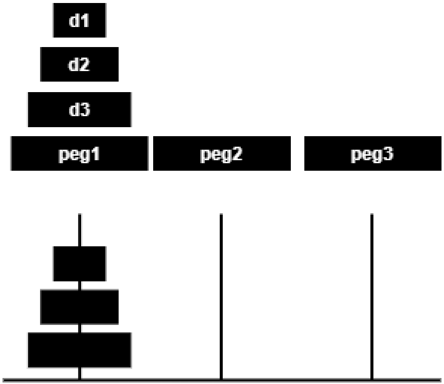
    - The Blocks World can have "counter-intuitive" states where for instance `holding(R,C)` and `ontable(C)` are true at the same time



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

Tower of Hanoi: Modeling

- "Depending on the formulation" $\Longrightarrow$ We need a formulation!
  - We begin with some modeling tricks!



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

Tower of Hanoi: Modeling (2)

- Domain

```
(: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))))
)
```

- Problem

```
 (: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))))
```

Tower of Hanoi: Number of States - How many are they?

- Domain

```
(:define (domain hanoi)
 (:requirements :strips)
 (:predicates (clear ?x) (on ?x ?y) (smal
 (:action move
   :parameters (?disc ?from ?to)
   :precondition (and (smaller ?to ?disc)
   :effect (and (clear ?from) (on ?disc ?
)
```

- Problem

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

Answer

Every complete 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!

The state is just a data structure
Every value combination is a state

Tower of Hanoi: Modeling alternatives

- Initial formulation

2^{78} combinations in total!

- 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!

- Suppose we get rid of "`clear`" (redundant):
  - Use more expressive planner
  - `(clear ?x)` $\Longrightarrow$ `(not (exists ?y) (on ?y ?x))`

2^{6*6} combinations of `on`
2^{36} combinations in total!

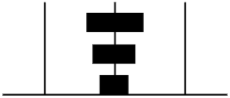
- Suppose we remodel "`on`":
  - `below_d1` $\in$ {`peg1,peg2,peg3,d2,d3`}
  - `below_d2` $\in$ {`peg1,peg2,peg3,d1,d3`}
  - `below_d3` $\in$ {`peg1,peg2,peg3,d1,d2`}

$5^{3} = 125$ combinations in total!

Why the extreme dependence on the formulation?

Model Dependence

- In all suggested formulations of the ToH, this is one possible state
  - Planners should not generate such states, but they still exists!



- In some formulations, states such as this exists
  - `(and (on peg1 peg2) (on d1 d2) (on d2 d1) (on d3 d3) ...)`
- In the last formulation example:
  - `below_peg1` does not exists
  - `below_d3` cannot be d3
  - (but we can still have circularities)

$$\texttt{below\_d1} \in \{\texttt{peg1}, \texttt{peg2}, \texttt{peg3}, \texttt{d2}, \texttt{d3}\}$$
$$\texttt{below\_d2} \in \{\texttt{peg1}, \texttt{peg2}, \texttt{peg3}, \texttt{d1}, \texttt{d3}\}$$
$$\texttt{below\_d3} \in \{\texttt{peg1}, \texttt{peg2}, \texttt{peg3}, \texttt{d1}, \texttt{d2}\}$$

Some formulations allow more "unintended states" than others!

Does the size of the state space matter?

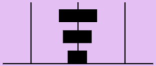
Reachability

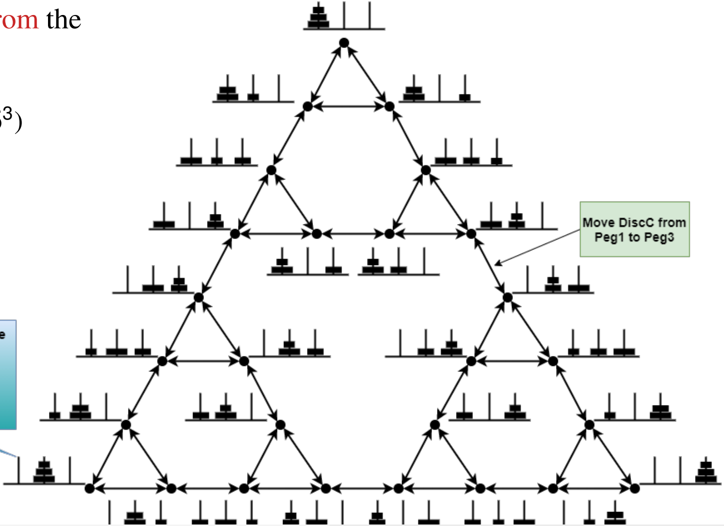
- Forward state space search:
  - Incrementally generate (only) reachable states
  - Many unreachable states?
    - More state variables $\Longrightarrow$ somewhat more expensive to generate/store a state
- Uninformed strategies (Depth/Breadth First, Dijkstra, ...)
  - No difference in what explored!
- 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: from initial state

- How many states are reachable from the given initial state using the given actions?
  - 27 out of 2^{78} (or of 2^{42}, 2^{36}, 5^{3})

The other states still exists in S!





Reachability: from somewhere

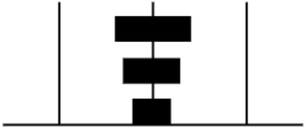
Important concept about reachability!

States are not inherently "reachable" or "unreachable"

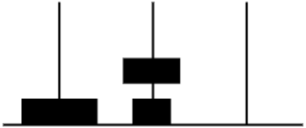
They can be reachable from a specific starting point!

Reachability: from "forbidden" states

- Suppose this was your initial state
  - Unreachable from state where "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!

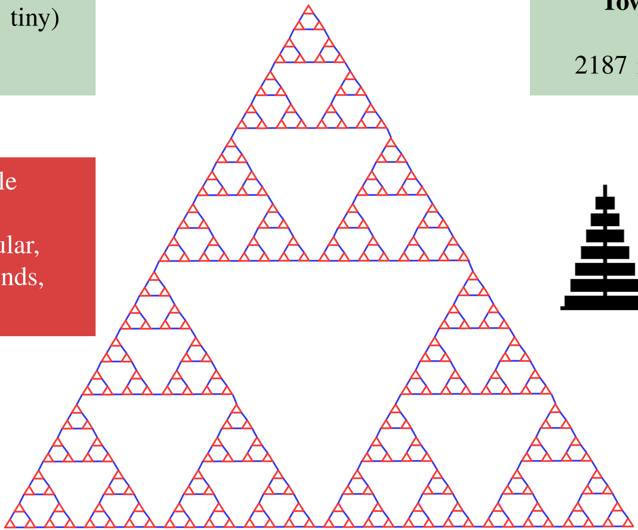


Start in physically realizable states $\Longrightarrow$ remain there (assuming correct operators)
Start somewhere else $\Longrightarrow$????

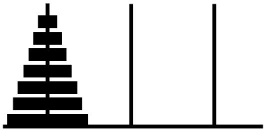
Reachability: Larger example

A larger (but still tiny) example...

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

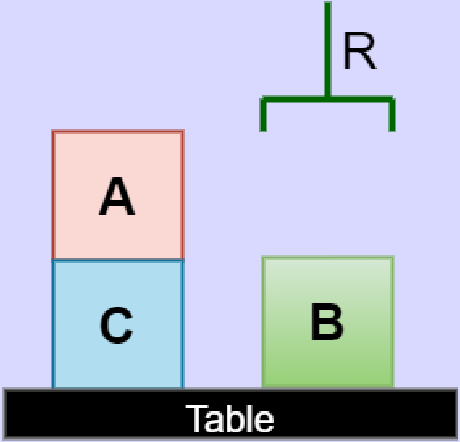


Tower of Hanoi
7 disks
2187 reachable states

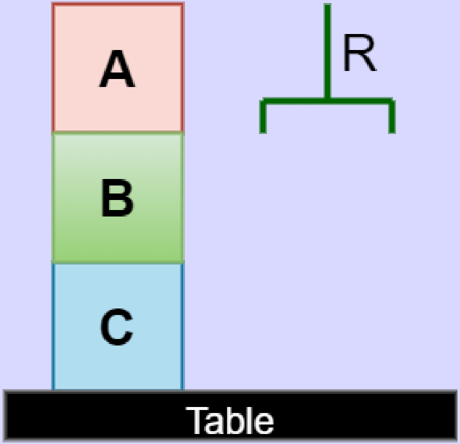


Blocks World

Initial state

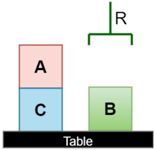


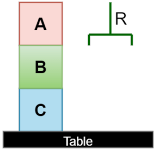
Goal state



Blocks World: Model

- We will generate classical sequential plans
  - One object type: `blocks`
  - A common blocks world version, with 4 operators
    - `(pickup ?x)` - take `?x` from the `table`
    - `(putdown ?x)` - puts `?x` from the `table`
    - `(unstack ?x ?y)` - take `?x` from on top of `?y`
    - `(stack ?x ?y)` - put `?x` on top of `?y`
  - Predicates used
    - `(on ?x ?y)` - block `?x` is on block `?y`
    - `(ontable ?x)` - `?x` is on `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)` → `(stack A B)`

Blocks World: Operator reference

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

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

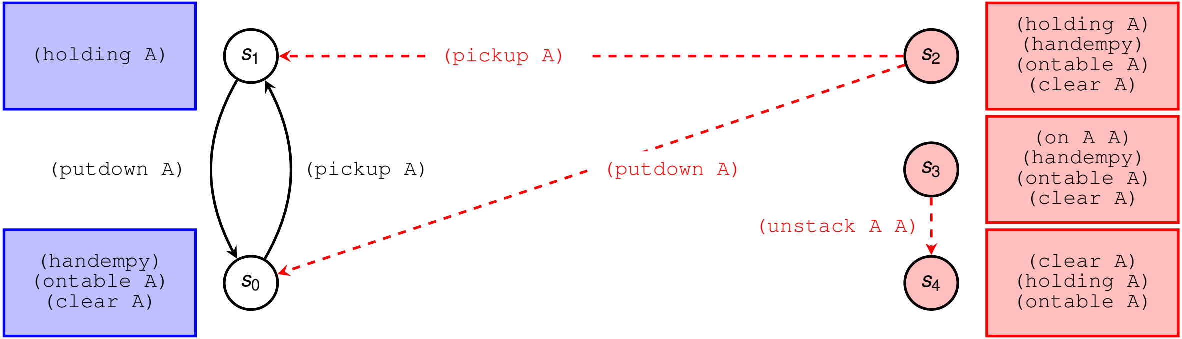
```
(:action putdown
  :parameters (?x)
  :precondition (and (holding ?x))
  :effect (and (ontable ?x)
               (clear ?x)
               (handempty)
               (not (holding ?x))))

(: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)))
```

Blocks World: Reachable State Space - 1 Block

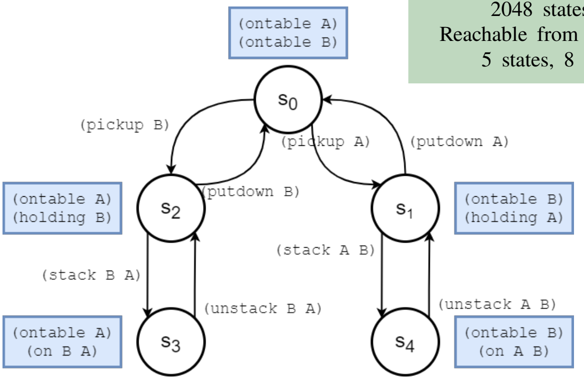
We assume we know the initial state
Let's see which states are reachable from there!

Start with s_0 = all blocks on the table

Many other states exists
but are not reachable
from s_0



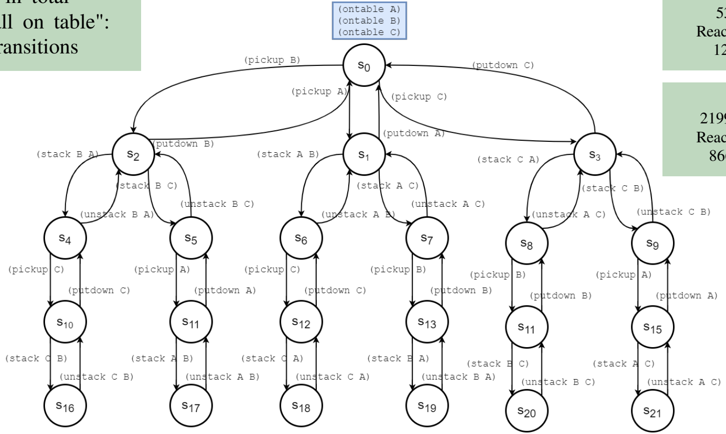
Blocks World: Reachable State Space - 2 Blocks



2048 states in total
Reachable from "all on table":
5 states, 8 transitions

Blocks World: Reachable State Space - 3 Blocks

524288 states in total
Reachable from "all on table":
22 states, 42 transitions



4 Blocks
536870912 states in total
Reachable from "all on table":
125 states, 272 transitions

5 Blocks
2199023255552 states in total
Reachable from "all on table":
866 states, 2090 transitions

Looking nice and symmetric...

Size: Blocks World - PDDL

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

Size: Blocks World - Reachable State Space Size 0-10

Blocks	Ground atoms	States	States reach from all on table	Trans. Reach. 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	36028797018963968	7057	18552
...				
9	109	649037107316853453566312041152512	8145730	25951122
10	131	2722258935367507707706996859454145691648	...	...

Blocks World: 5 Blocks

Standard PDDL

$$2^{n^2+3n+1}$$

2199023255552 states
(reachable and unreachable)

866 reachable

Modified PDDL

Omit `(ontable ?x)`,`(clear ?x)`
In *physically achievable* states, can be deduced from `(on ?x ?y)`, `(holding ?x)`

$$2^{n^2+n+1}$$

2147483648 states
(reachable and unreachable)

866 reachable

Blocks World: Formulations

- Example: Blocks World with 5 blocks
  - 2199023255552 or 2147483648 states in the standard predicate representation
- But in all 866 states reachable from all-on-table
  - Any state satisfies exactly one of the following - a clique:
    - `(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
    - `(on E A)` - Below E

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

Useful in some situations such us PDB heuristics

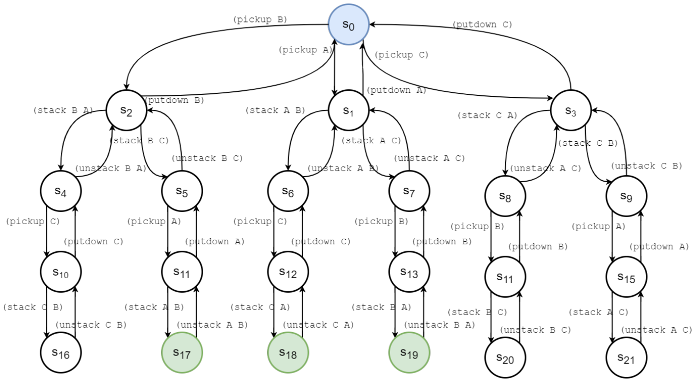
  - Remove those facts, introduce state variables (same for other blocks):
    - `aboveA` ∈ {`gripper`, `nothing`, `B`,`C`,`D`,`E`}
  - Result: $(n+1)^n 2^{n+1} = 497664$ states, 866 reachable!

Dock Worker Robots: Example

- Example 1: 1 location, 2 piles, 1 robot, 2 cranes, 2 containers
  - 2^{35} states
  - 16 states reachable and 32 transitions (given one particular initial state)
- Example 2: 2 location, 4 piles, 1 robot, 2 cranes, 2 containers
  - 2^{65} states
  - 100 states reachable and 332 transitions (given one particular initial state)
- Example 3: 2 location, 4 piles, 1 robot, 2 cranes, 3 containers
  - 2^{83} states
  - 756 states reachable and 2916 transitions (given one particular initial state)
- Example 4: 2 location, 4 piles, 1 robot, 2 cranes, 4 containers
  - 2^{103} states
  - 6192 states reachable and 25968 transitions (given one particular initial state)
- Example 5: 2 location, 4 piles, 1 robot, 2 cranes, 6 containers
  - 2^{149} states
  - 542880 states reachable and 2486880 transitions (given one particular initial state)
- Example 6: 3 location, 6 piles, 1 robot, 3 cranes, 3 containers
  - 2^{207} states
  - 1313280 states reachable and 6373440 transitions (given one particular initial state)

FORWARD STATE SPACE SEARCH

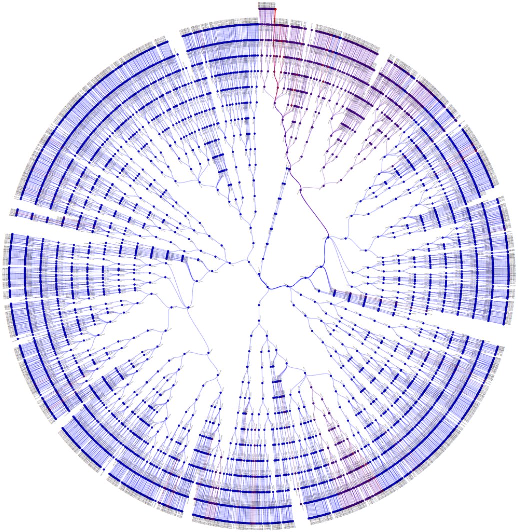
Find a path in the forward state space from the initial state (node) to to any goal state



Many names used in the literature: Forward search, Forward-chaining search, Forward state space search, Progression, ...

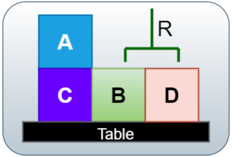
Forward State Space Search: Do not pre-compute!

- The planner is not given a complete pre-computed search graph!
- Usually it is too large!!!
- $\Longrightarrow$ Generate as we go ...
  - hoping the entire graph is not needed!



Forward State Space Search: Initial State!

- The agent 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) }

Forward State Space Search: Successors!

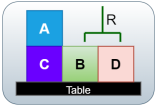
- Given any open search node (to be selected by a *strategy*)...

```
{(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 B), (clear D), (ontable D), (handempty)` // preconditions // satisfied

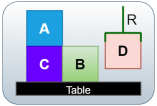
      Effects: `(not (ontable D)), (not (clear D)), (holding D), (not (handempty))`
- This generates new new reachable states ...

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



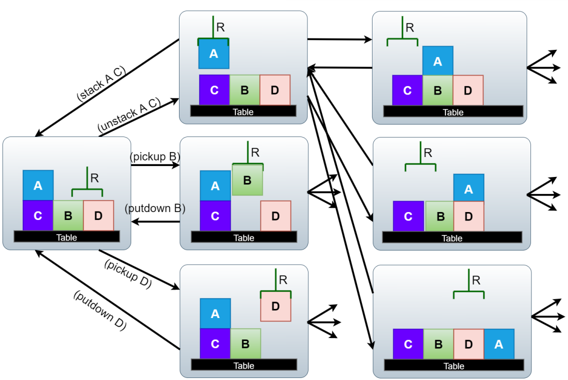
(pickup D)

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



Forward State Space Search: Step by Step

- 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 initial state and goal state



FORWARD STATE SPACE SEARCH: PLANNING AS SEARCH

function SEARCH(problem)
    initial-node ← MAKE-INITIAL-NODE(problem) → *[2]*
    open ← {initial-node}
    while (open ≠ ∅) **do**
        node ← SEARCH-STRATEGY-REMOVE-FROM(open) → *[6]*
        if IS-SOLUTION(node) **then** → *[4]*
            return EXTRACT-PLAN-FROM(node) → *[5]*
        end if
        for each newnode ∈ SUCCESSORS(node) **do** → *[3]*
            open ← open ∪ {newnode}
        end for
    end while
    return Failure → *Expanded the entire search space without finding a solution*
end function

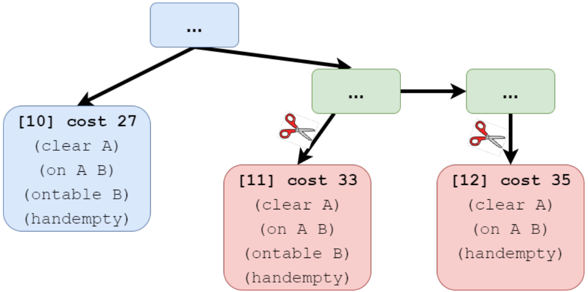
Forward State Space Search: Planning as Search

```
function SEARCH(problem)
    initial-node ← ⟨s_0, ε⟩                                   → [2] Nodes contain the plan to reach the node itself!
    open ← {initial-node}
    while (open ≠ ∅) do
        node = ⟨s, π⟩ ← SEARCH-STRATEGY-REMOVE-FROM(open)      → [6]
        if IS-SOLUTION(node) then                               → [4] check goal formula in state s
            return π                                            → [5]
        end if
        for each a ∈ A such that γ(s, a) ≠ ∅ do                 → [3]
            {s′} ← γ(s, a)                                      → Fwd search: reach in one step = reach by one action application
            π′ ← append(π,a)
            open ← open ∪ {⟨s′, π′⟩}
        end for
    end while
    return Failure                                              → Expanded the entire search space without finding a solution
end function
```

This algorithm is always sound! Completeness depends on the strategy!

Note: Technically, this algorithms search the space of ⟨ state,plan ⟩ pairs: still called state space search!

Forward State Space Search: Pruning



Reach a more expensive node with the same state
$\Longrightarrow$ can prune (discard the node without expanding)

If preconditions and goals are positive, if I reach a node with a *subset* of the facts
$\Longrightarrow$ can prune

Notice that...

If we allow negative preconditions (E.g. `(not (ontable B))`) $\Longrightarrow$ an action may be applicable in a subtree of [12] but not under [11] $\Longrightarrow$ node cannot be pruned, and subtree shall be investigated!

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`.

[4] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019. doi: 10.2200/S00900ED2V01Y201902AIM042. URL `https://doi.org/10.2200/S00900ED2V01Y201902AIM042`.