Course "Automated Planning: Theory and Practice"
Chapter 05: The Backward Search 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

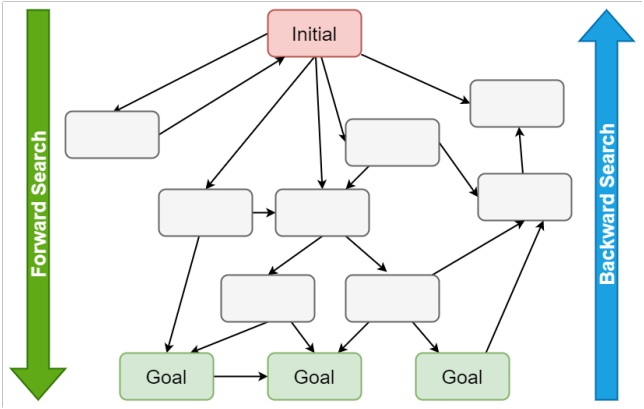
Last updated: Sunday 5th October, 2025

Terms of Use and Copyright

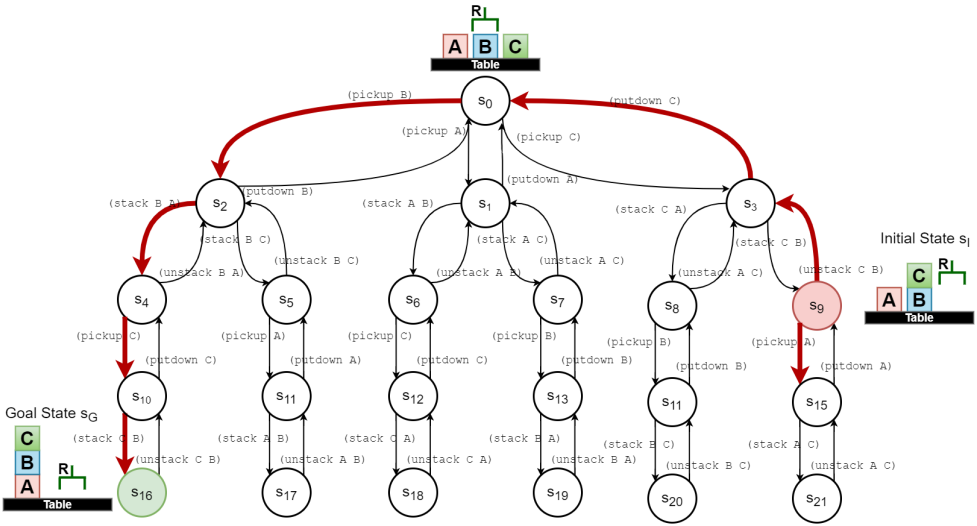
The material (text, figures) in these slides is authored by Jonas Kvarnström and Marco Roveri.

Introduction

- Classical Planning: find a path in a finite graph
  - We have seen how to search forward
  - Can we search backward? How?



Blocks World, 3 blocks - searching forward



Blocks World, 3 blocks - searching backward

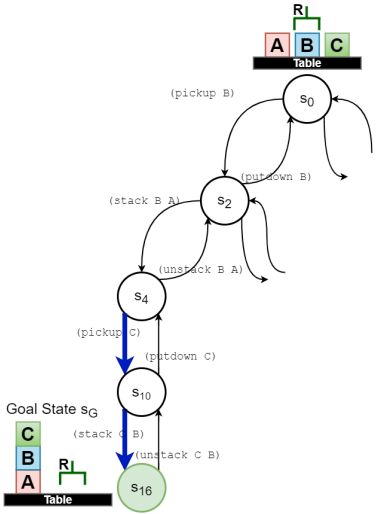
1: Execution should pass s_4 ...

2: Execute `(pickup C)` ...

3: Pass s_{10} ...

4: Execute `(stack C B)` ...

5: ... and end up in s_{16}



5: Pass s_4 ...

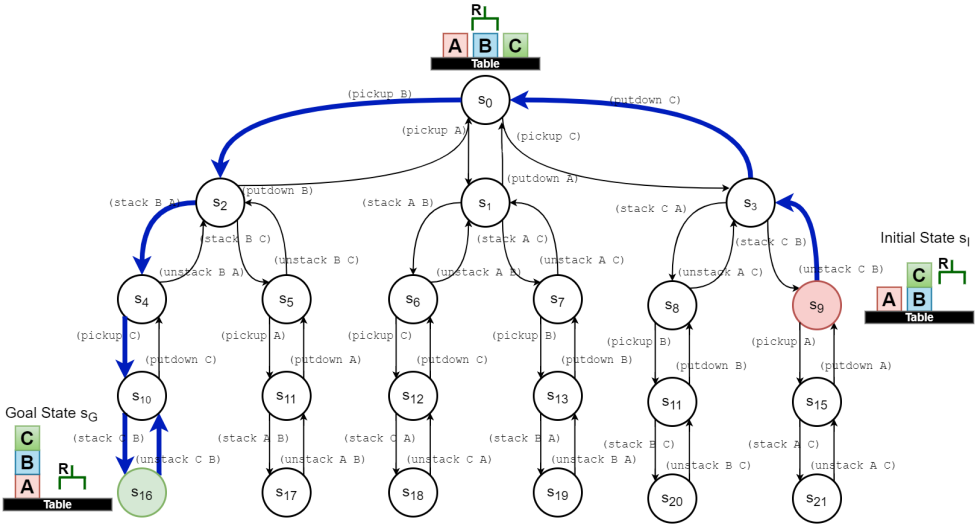
4: Previous action could be `(pickup C)` ...

3: Pass s_{10} ...

2: Previous action could be `(stack C B)` ...

1: Planning starts in s_{16}

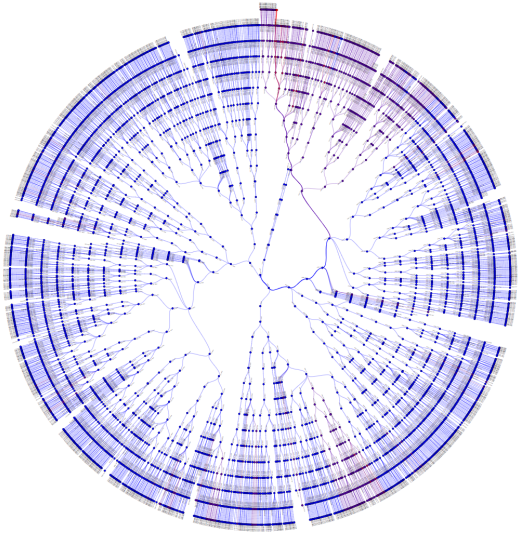
Backward Search



Seems simple, ...but there are complications...

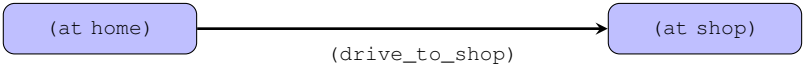
Backward Search: Complication 1

- The graph is not pre-computed!
  - Must be expanded dynamically, starting in the *goal* states
- Would require the inverse of $\gamma(s, a)$: $\gamma^{-1}(s, a)$

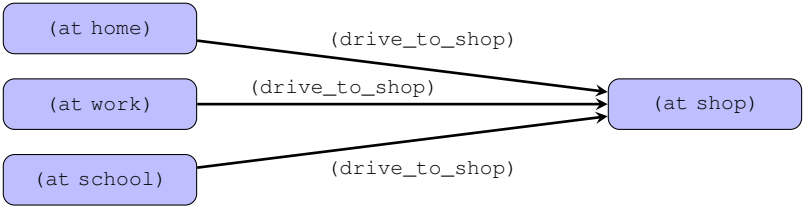


Backward Search: Complication 2

- Though we have determinism in the forward search



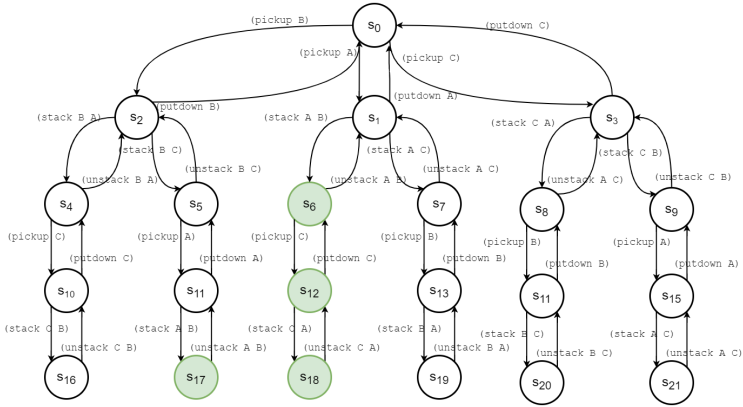
- ... this is not the case in the backward direction!



- Compute $\gamma^{-1}(\{\texttt{(at home)}\}, \texttt{(drive\_to\_shop)})$
  - If we want to end up in $\{\texttt{(at shop)}\}$,
    what set of states could we be in before `(drive_to_shop)`?

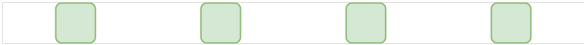
Backward Search: Complication 3

- We generally have multiple goal states - to start searching in...
  - Goal: `(on A B)`

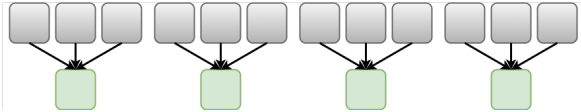


Backward Search: Complication 2-3 (combinations)

- Want to end up in one of these goal states

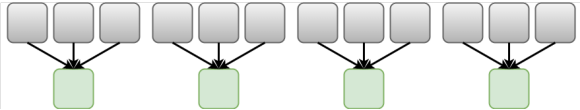


- Even if we say the last action is `(drive_to_shop)`,...
  we could have started in any of these states



- Given initial state + forward plan [`(drive_to_shop)`]
  - One possible next state
- Given goal state(s) + backward plan [`(drive_to_shop)`]
  - Many possible previous states

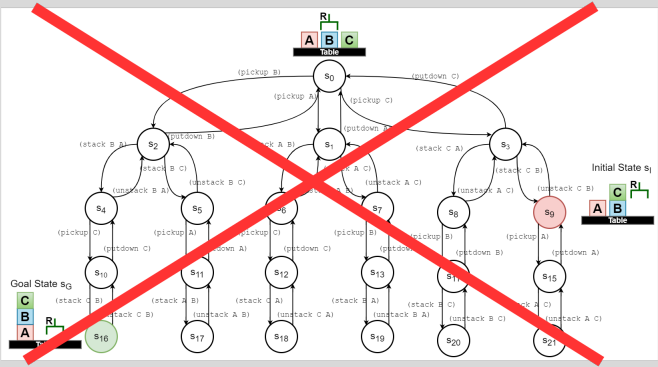
- Main challenge: A set of possible "current" states
  - Can't store and process each state separately



- Classical representation:
  - Goal: set of literals that should hold, representing multiple states: $g = \{(\texttt{on A B}), \neg(\texttt{on C D})\}$
    - A should be on B, and C should not be on D
    - We don't care if the blocks are clear/ontable or not
      $\Longrightarrow$ if we cared, that would have been specified!

Goal Space ≠ State Space

Backward Search uses goal space



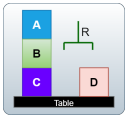
Will not construct this graph $\Longrightarrow$ use $\gamma^{-1}(g, a)$, not $\gamma^{-1}(s, a)$

If you achieve conditions in $\gamma^{-1}(g, a)$, then executing a will achieve g

How can we construct a goal space begining with an "initial goal"?

Goal specification

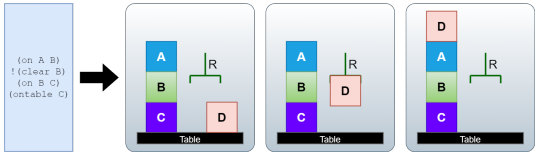
- Assuming the goal is:



- What's the actual goal specification?
  - We could specify a complete (i.e. unique state)
    - $g =$ `{(clear A),(on A B),(on B C),(ontable C),(clear D),(ontable D),(handempty),¬(clear B),¬(on A A), ...}`
  - Or we can just specify "the important" things (what we expect to hold)
    - $g =$ `{(clear A),(on A B),(on B C),(ontable C),(clear D),(ontable D)}`
    - Specify all positions: given a *physically achievable initial state*, other facts follow implicitly

Goal specification

- Usually we do not care about all facts (directly or indirectly)!
  - Ignore location of *D*
  -



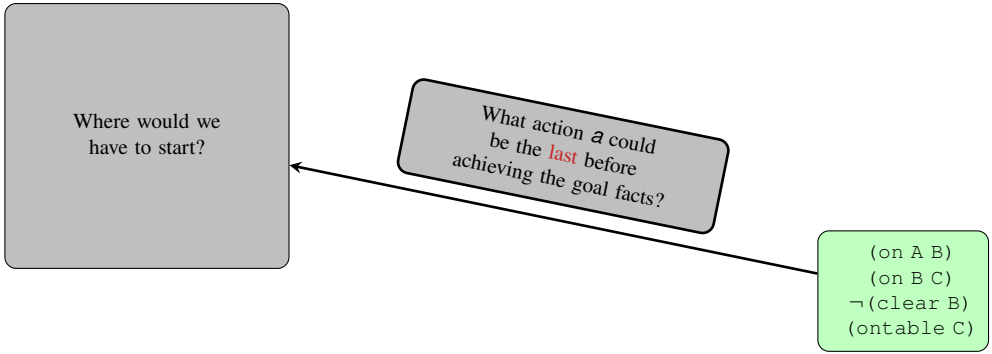
Forward Planning: action applicability

Which actions could we execute?

Backward Planning: action relevance

Which actions could achieve part of the goal?

Backward Search: Relevance



Backward Search: Relevance (cont.)

NO!
It achieves
`(clear ?t) = (clear B)`
The goal requires
$\neg$`(clear B)`

$\Longrightarrow$ Destroys part of the goal!

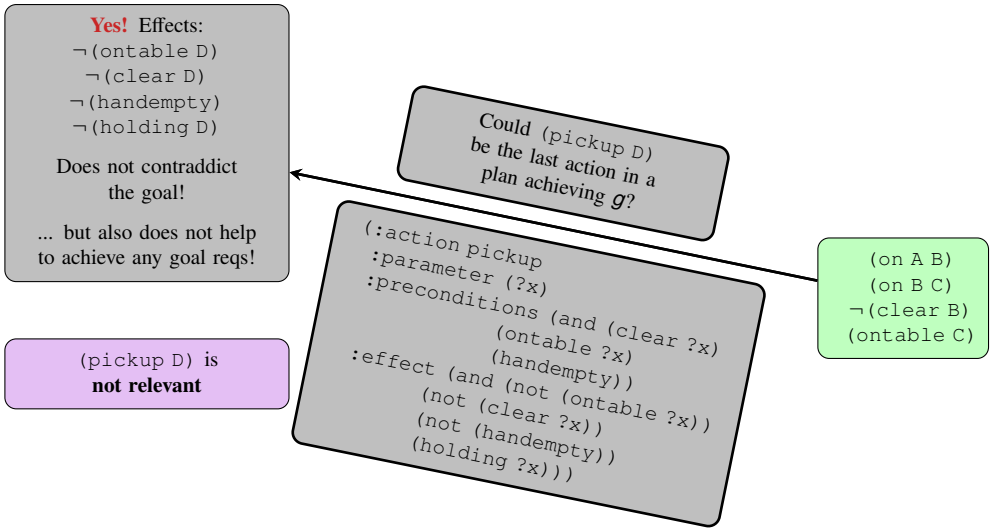
`(stack B C)` is **not relevant** (also *impossible*, but this is included in relevance!)

Could `(stack B C)` be the last action in a plan achieving g?

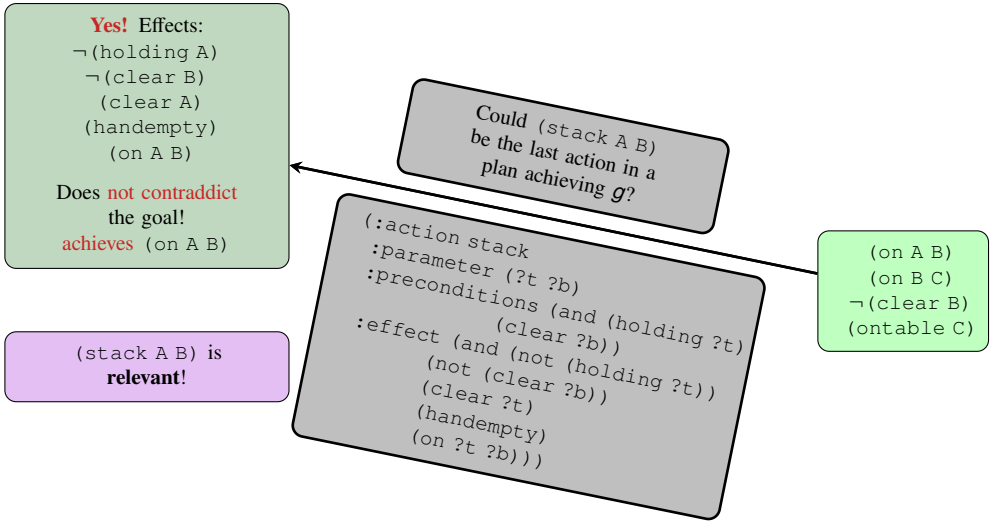
```
(:action stack
 :parameter (?t ?b)
 :preconditions (and (holding ?t)
                     (clear ?b))
 :effect (and (not (holding ?t))
              (not (clear ?b))
              (clear ?t)
              (handempty)
              (on ?t ?b)))
```

`(on A B)`
`(on B C)`
$\neg$`(clear B)`
`(ontable C)`

Backward Search: Relevance (cont.)



Backward Search: Relevance (cont.)



Backward Search: Summary (so far)

Forward search over states $s = \{\text{atom}_1,\ldots,\text{atom}_N\}$

- a is applicable to *current state* s iff:
  - $precond^+(a) \subseteq s$ and — Positive conditions are present
  - $s \cap precond^-(a) = \emptyset$ — Negative conditions are absent

Backward search over set of literals $g = \{\text{lit}_1,\ldots,\text{lit}_N\}$

- a is relevant for *current goal* g iff:
  - $g \cap effects(a) \neq \emptyset$ and — Contribute to the goal: add needed pos/neg literals
  - $g^+ \cap effects^-(a) = \emptyset$ and
  - $g^- \cap effects^+(a) = \emptyset$ — Do not destroy any goal literal

When an action has been selected:

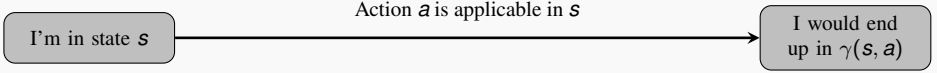
Forward Planning: Progression!
What will be true after executing a?

Backward Planning: Regression!
What must be achieved before executing a?

Backward Search: Progression and Regression

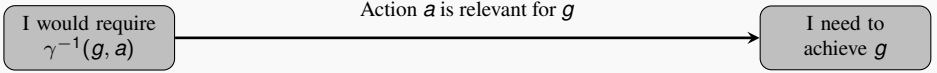
Forward search over states $s = \{\text{ATOM}_1,\ldots,\text{ATOM}_N\}$

- Progression: $\gamma(s, a) = \{s \setminus effects^-(a) \cup effects^+(a)\}$



Backward search over set of literals $g = \{\text{LIT}_1,\ldots,\text{LIT}_N\}$

- Progression: $\gamma^{-1}(g, a)$ =???



Backward Search: Regression

$g' = \gamma^{-1}(g, (\texttt{stack A B}))$

What facts g' would we require *before executing* a, so that for every state s satisfying g':
1) a is executable in s
2) $g \subseteq \gamma(s, a)$?

What action a could be the last before achieving the goal facts?

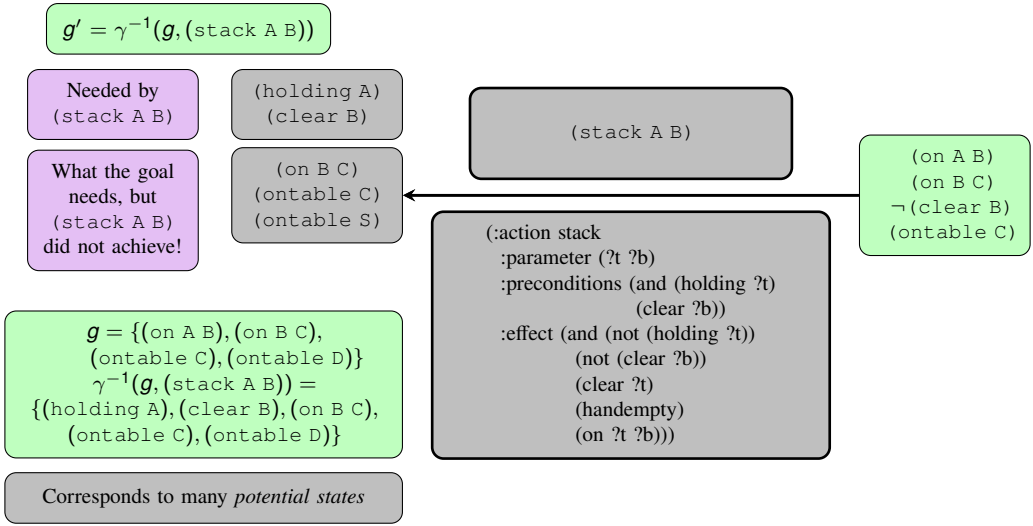
(on A B)
(on B C)
¬(clear B)
(ontable C)

Subset: It is OK to achieve more than required!

$g = \{(\texttt{on A B}), (\texttt{on B C}), (\texttt{ontable C}), (\texttt{ontable D})\}$

$\gamma(s, a) = \{(\texttt{on A B}), (\texttt{on B C}), (\texttt{ontable C}), (\texttt{ontable D})$
$(\texttt{clear A}), (\texttt{clear D}), (\texttt{handempty})\}$

Backward Search: Regression



Backward Search: Regression - Formalization

All goals except $effects(a)$ must alredy have been true

$precond(a)$ must have been true so that a was applicable

$$\gamma^{-1}(g, a) = ((g \setminus effects(a)) \cup precond(a))$$

representing

$\{s | a$ is applicable to s and $\gamma(s, a)$ satisfies $g\}$

Backward regression: Which states could I start from?

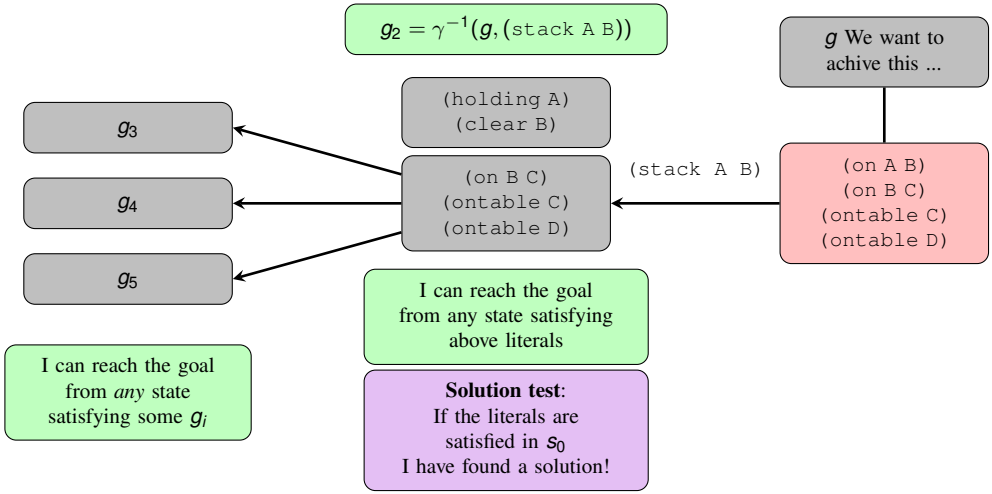
Works for

Classical goals (already sets ground literals)

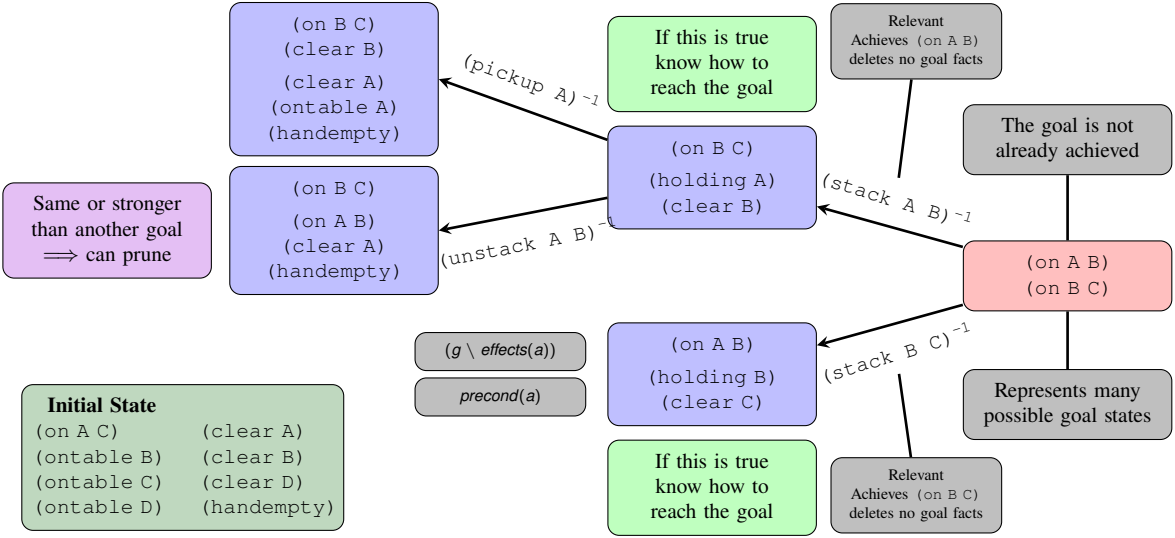
Classical effects (conjunction of literals)

Classical preconditions (conjunction of literals)

Backward Search: Keep regressing



Backward Search: Example

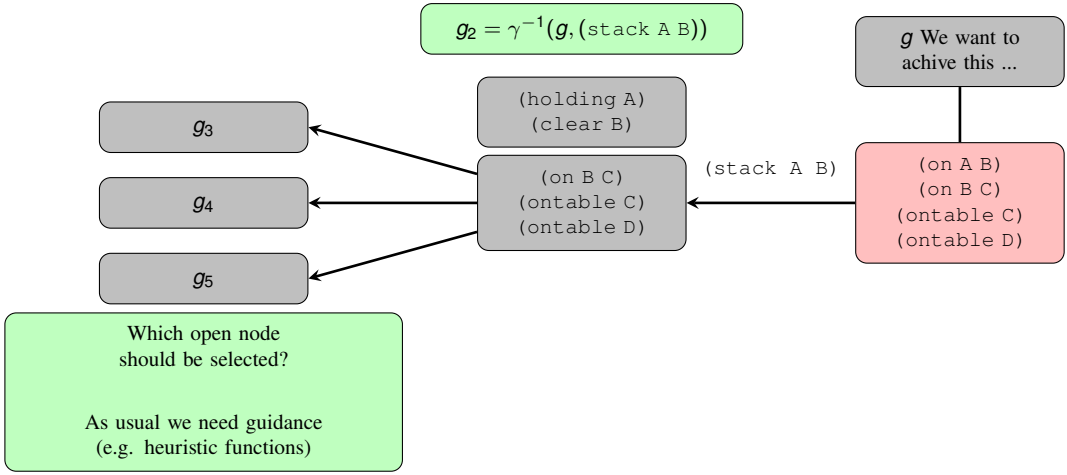


When we do select actions:

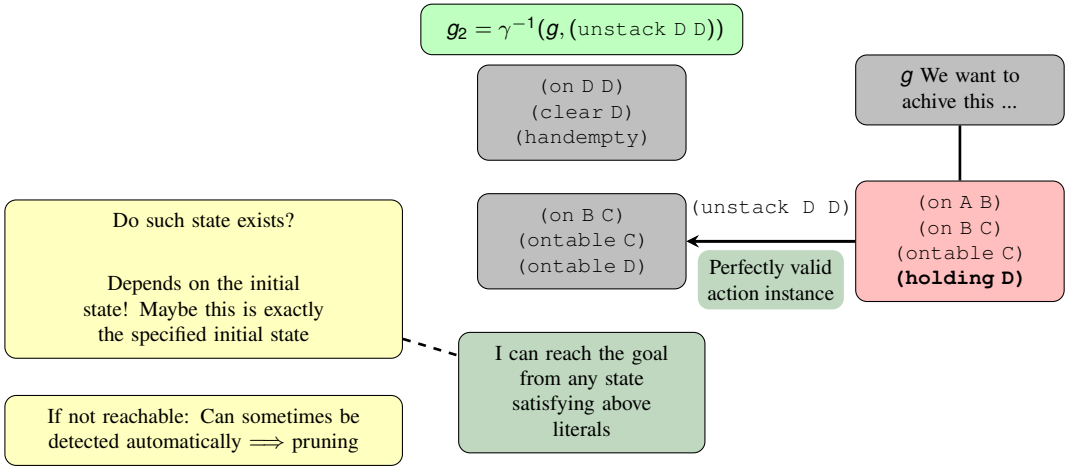
Forward planning:
Want the resulting state to be closer to the goal!

Backward planning:
Want the resulting goal to be closer
to what the initial state can satisfy!

Backward Search: Needs guidance

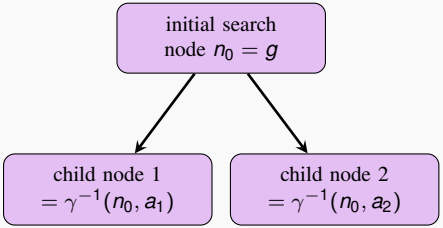


Backward Search: New goal achievable



Goal Space

The backward goal space for backward planning regression



2) **Initial search node**:
Corresponds directly to the specified goal

3) **Branching rule**:
For every action a relevant to the goal g of a node n, generate the goal $\gamma^{-1}(g, a)$
Represents the set of states where $\gamma(s, a)$ satisfies g

4) **Solution criterion**:
The goal of the node is satisfied in the initial state

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

Planning as Search (recap)

```
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
```

Backward Search: Instantiated Algorithm

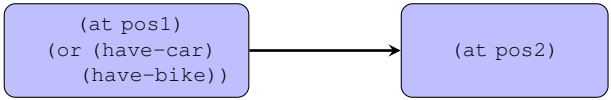
```
function SEARCH(problem)
    initial-node ← ⟨goal, ϵ⟩                                  → [2]
    open ← {initial-node}
    while (open ≠ ∅) do
        node = ⟨g, π⟩ ← SEARCH-STRATEGY-REMOVE-FROM(open)     → [6]
        if IS-SOLUTION(node) then                             → [4] Check goal formula in state s_0
            return π                                          → [5]
        end if
        for each a ∈ A relevant to g do                       → [3]
            g' ← γ^{-1}(g, a)
            π' ← PREPEND(a,π)
            open ← open ∪ {⟨g', π'⟩}
        end for
    end while
    return Failure                                            → Expanded the entire search space without finding a solution
end function
```

Backward and Forward Search: Expressivity

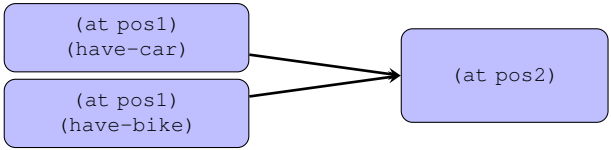
- Suppose we have disjunctive preconditions - simple in forward planning

```
(:action travel
  :parameters (?from ?to - location)
  :precondition (and (at ?from) (or (have-car) (have-bike)))
  :effect       (and (at ?to) (not (at ?from))))
```

- How do we apply such action backwards?
  - More complicated disjunctive goals to achieve?



  - Additional branching?



Similarly for existentials

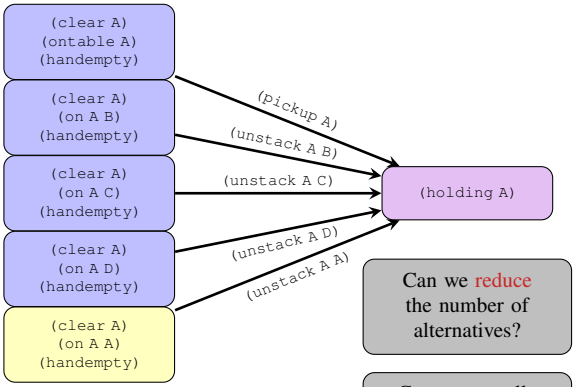
```
(exits ?block (on ?block A))
```

One branch per possible value

Some other extensions are less straightforward in backward search (but still possible!)

Lifted Search: Motivations

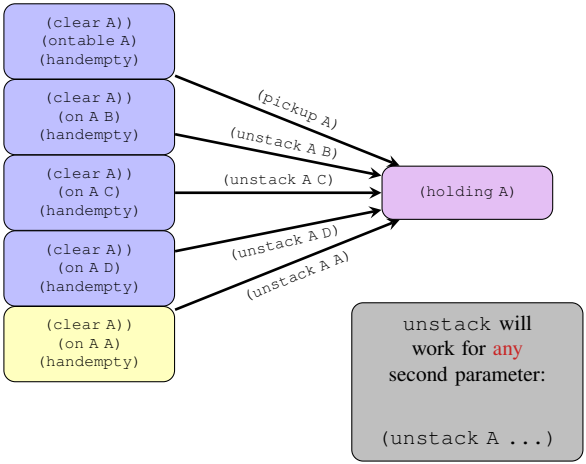
- High-branching-factors: potential problem in any search!




```
(: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 (holding ?top)
                (clear ?below)
                (not (clear ?top))
                (not (handempty))
                (not (on ?top ?below))))
```

Lifted Search: Observations

- High-branching-factors: potential problem in any search!




```
(: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 (holding ?top)
                (clear ?below)
                (not (clear ?top))
                (not (handempty))
                (not (on ?top ?below))))
```

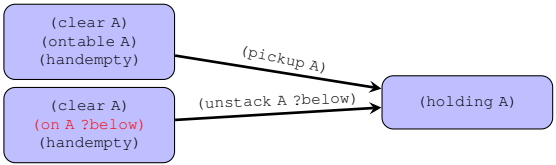
Lifted Search: General Idea

- Instantiate parameters that are "bound" by the goal
  - For `(pickup ?x)` to achieve `(holding A)`, we *must* have `?x = A`
- Keep other parameters uninitialized
  - For `(unstack ?top ?below)` to achieve `(holding A)`, we *must* have `?top = A`
  - We don't care about `?below`, so we don't give it a value: use `(unstack A ?below)`
- Not *ground* $\Longrightarrow$ "lifted"!

Must extend *relevance* for "pattern matching": Unification

Suppose `(on A B)` is true initially, or made true by an action A

Goal requires `(on A ?below)`
OK: `?below = B`



Only two new nodes to keep track of!

Applicable to other types if planning – we will see later!

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.