Linköping University

Automated Planning

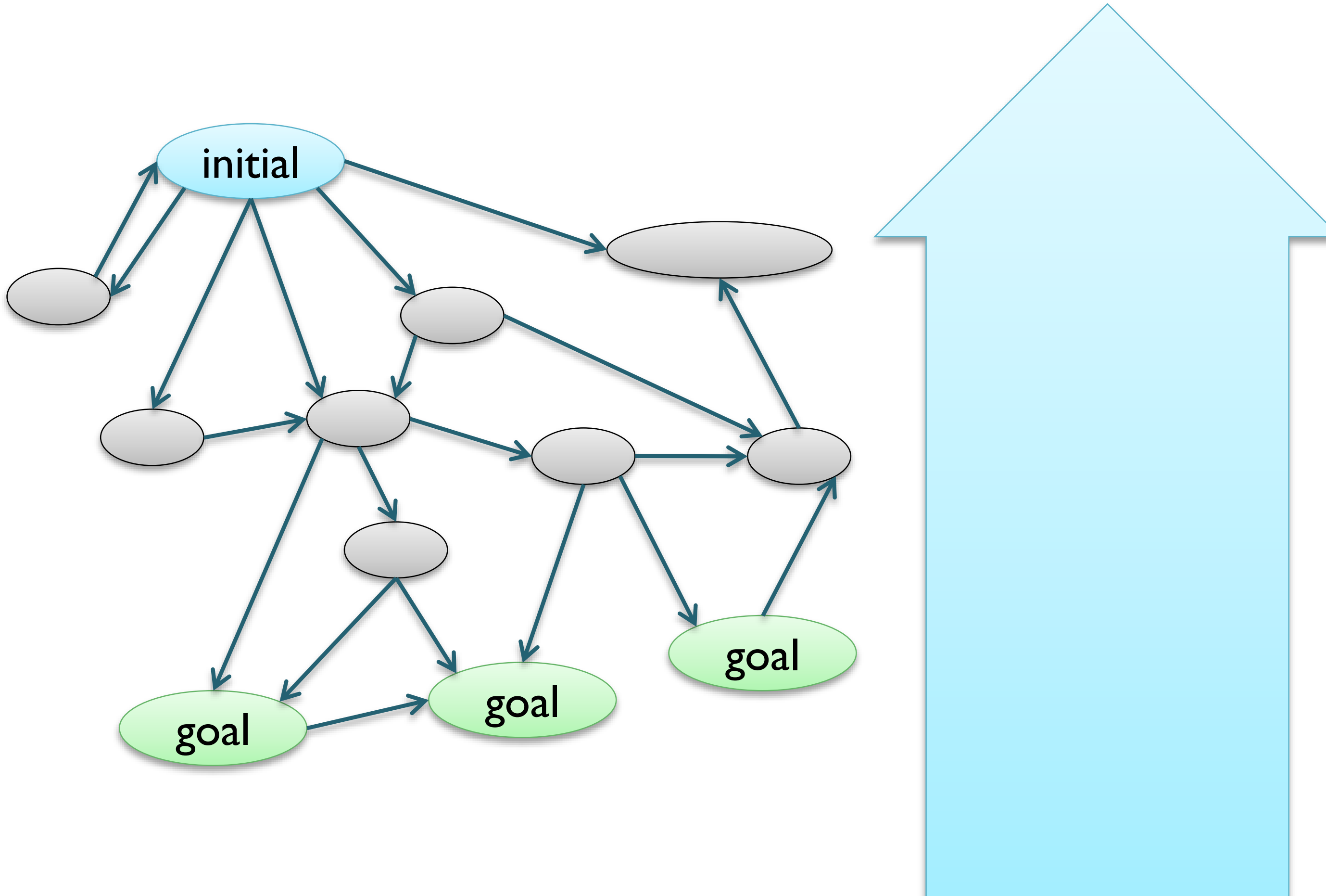
The Backward Goal Space

Jonas Kvarnström
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2019

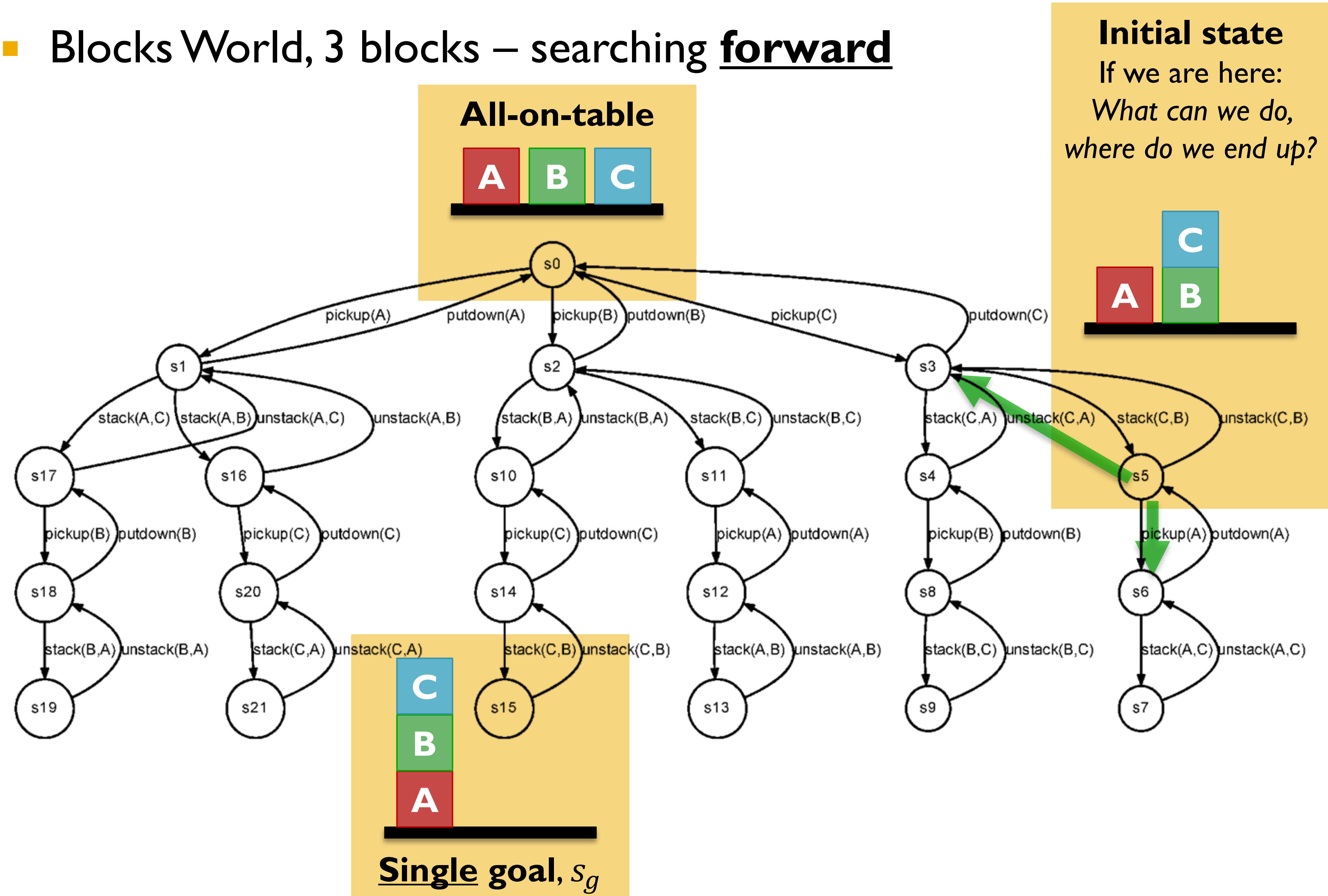
Intro

- Classical Planning: **<u>Find a path in a finite graph</u>**
  - We searched **<u>forwards</u>**
  - Can we search **<u>backwards</u>**? How?



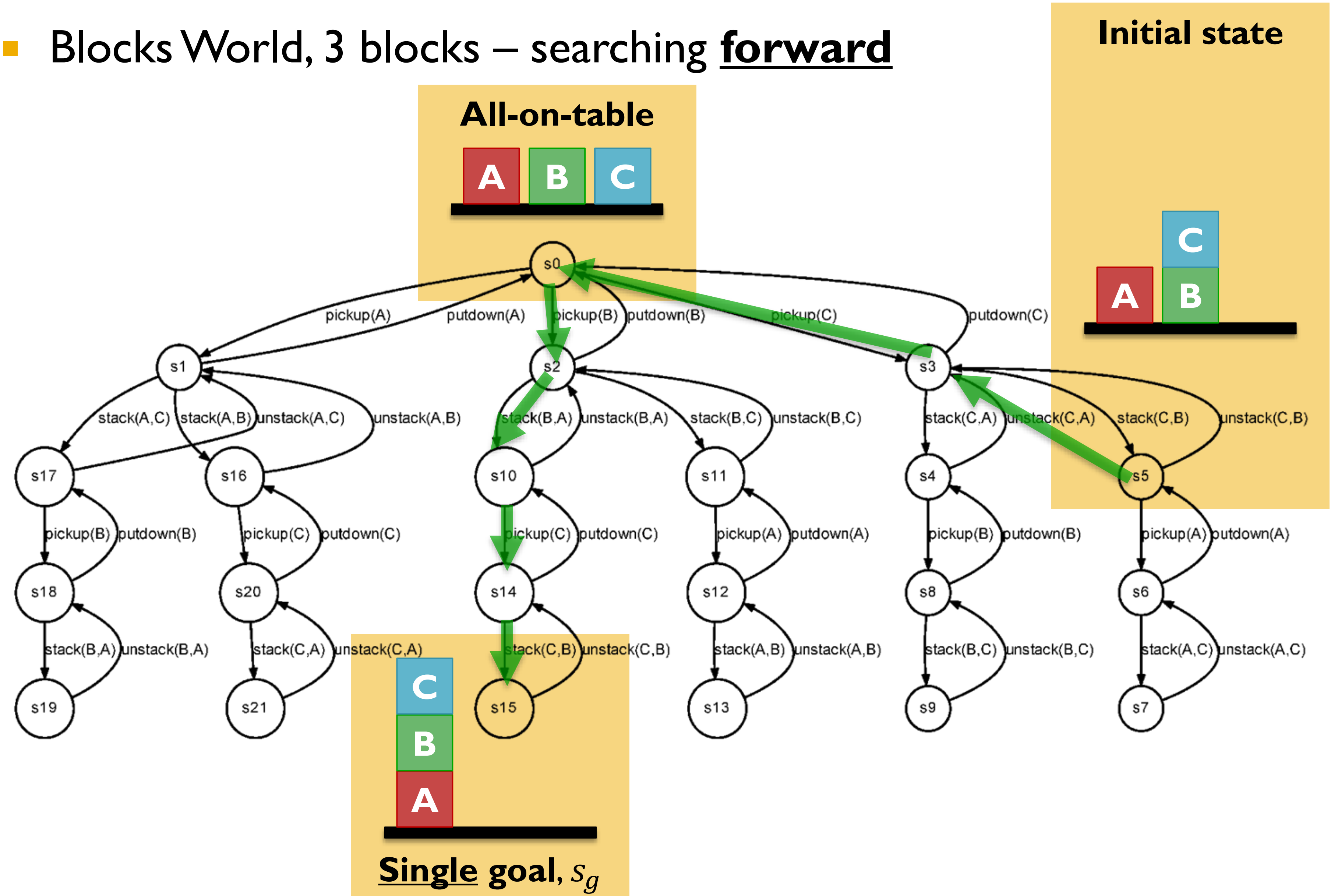
Contrast: Forward Search

- Blocks World, 3 blocks – searching **forward**



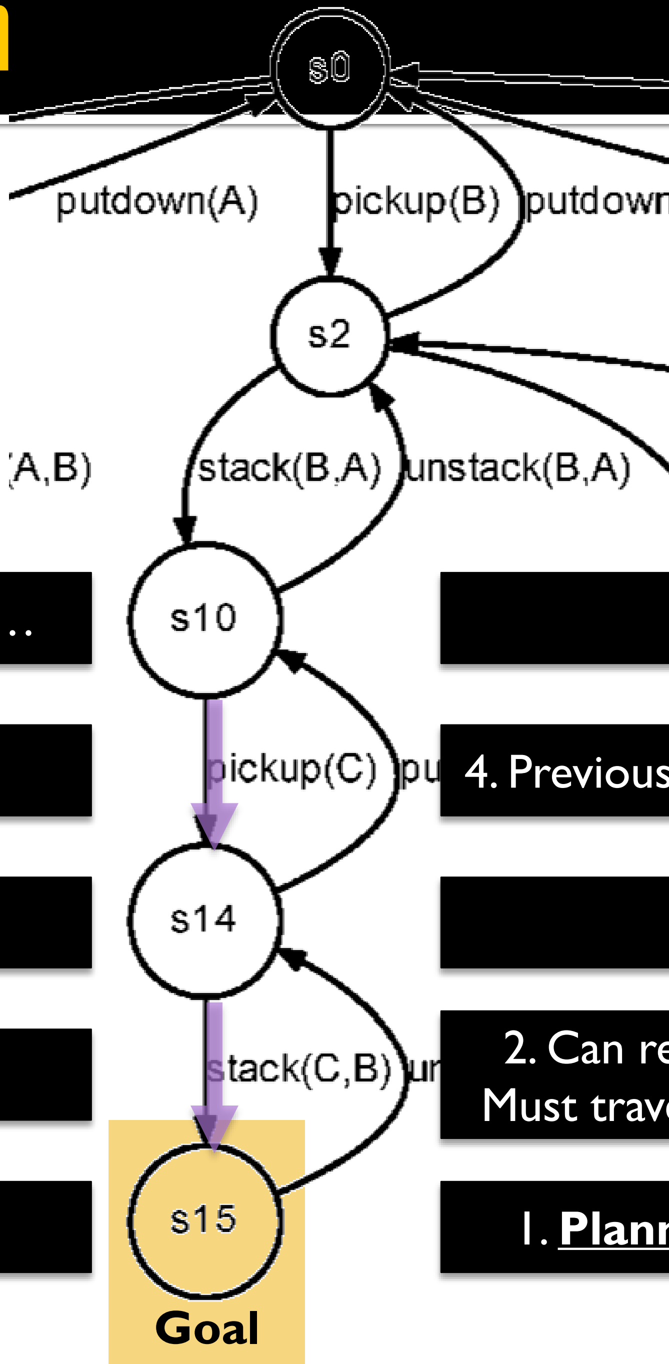
Contrast: Forward Search (2)

- Blocks World, 3 blocks – searching **forward**



Backward Search

- Must traverse edges backwards!



1. **Execution** should pass s10…

2. Execute pickup(C)…

3. Pass s14…

4. Execute stack(C,B)…

5. …and end up in s15

5. Pass s10…

4. Previous action could be pickup(C)

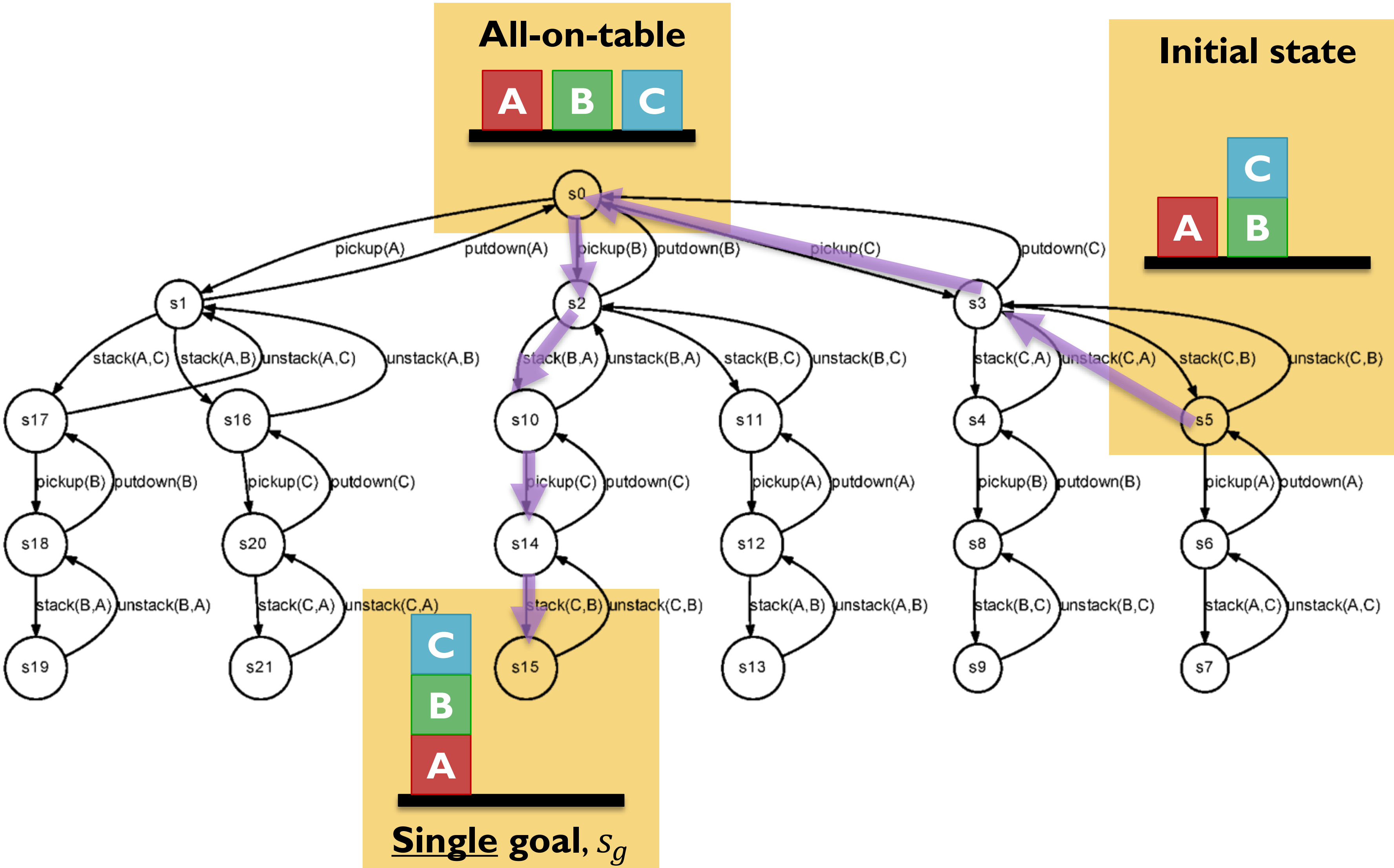
3. Pass s14…

2. Can reach s15 using stack(C,B)
Must traverse the edge ***backwards!***

1. **Planning** must start in s15…

Backward Search

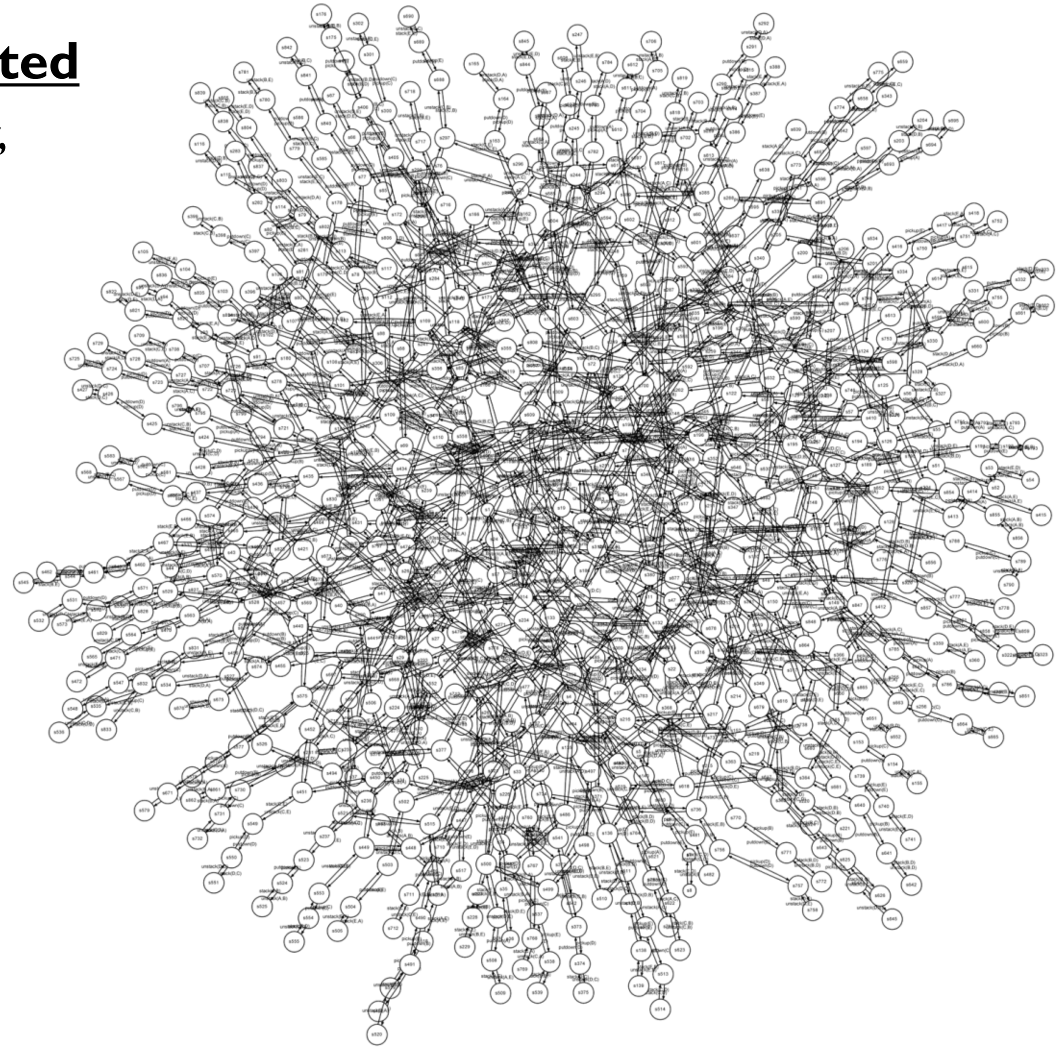
- Searching **backward**



Seems simple,
but there are complications…

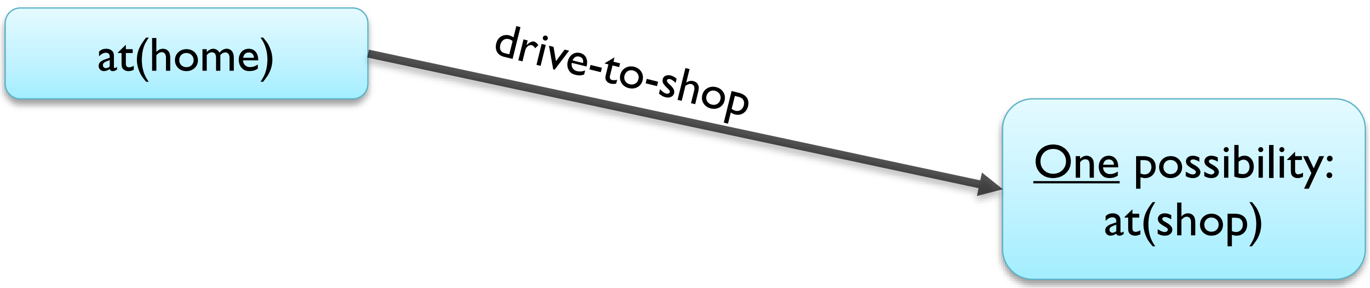
Backwards Search: Complication 1

- Complication 1:
  - **<u>The graph isn't precomputed</u>**
    - Must be expanded dynamically, starting in the *goal*
  - Would require an *inverse* of $\gamma(s, a)$: $\gamma^{-1}(s, a)$

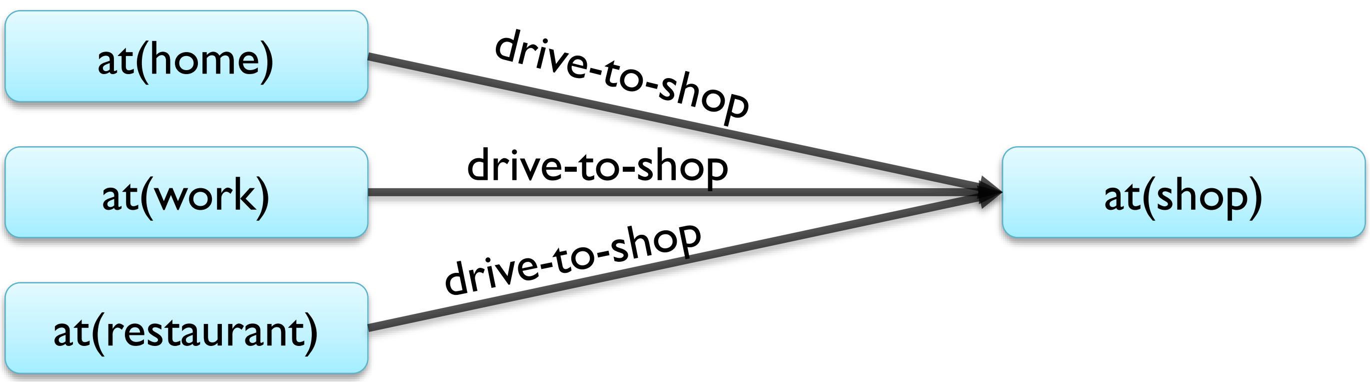


Backwards Search: Complication 2

- Complication 2:
  - Though we have **determinism** in the **forward** direction…



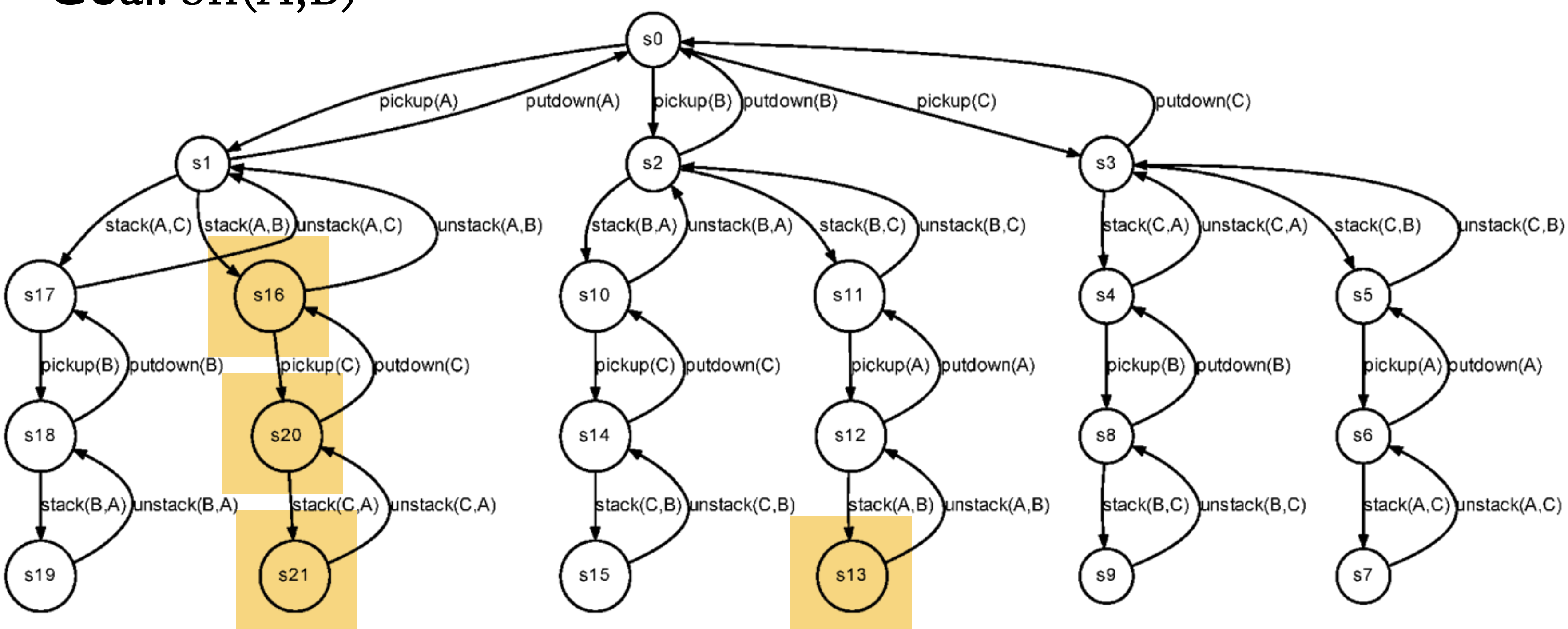
  - …this isn't the case in the **backward** direction!



  - Compute $\gamma^{-1}(\{at(shop)\}, \text{drive}-\text{to}-\text{shop})$:
    - If we want to **end up at(shop),**
      what **set of states** could we be in **before drive-to-shop**?

Backwards Search: Complication 3

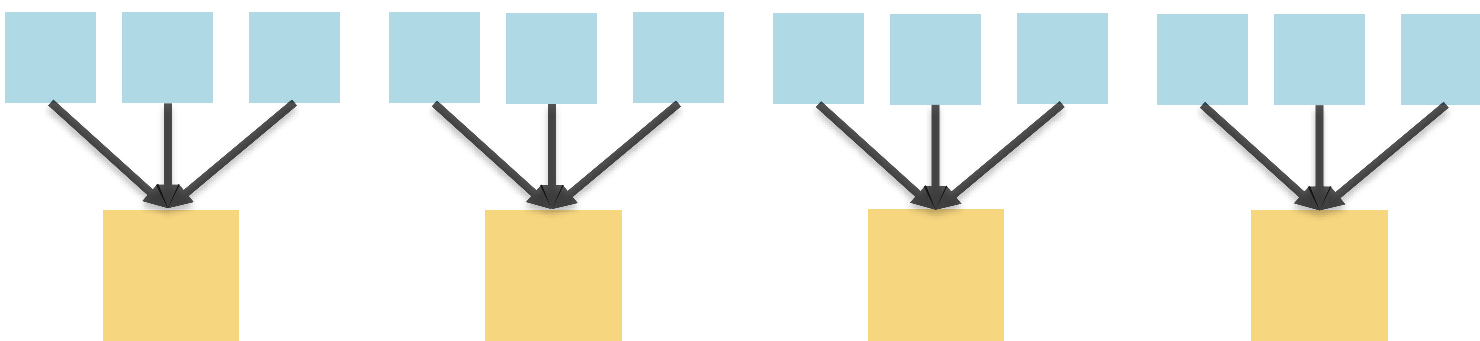
- Complication 3:
  - We generally have **<u>multiple goal states</u>** – to **start** searching in…
- Goal: on(A,B)



Backwards Search: Combinations

- Complications 2+3 combined:
  - Want to end up in one of these goal states (”at the shop”)

  - Even if we say the last action had to be **<u>drive-to-shop</u>**, we could have started in any of these states:

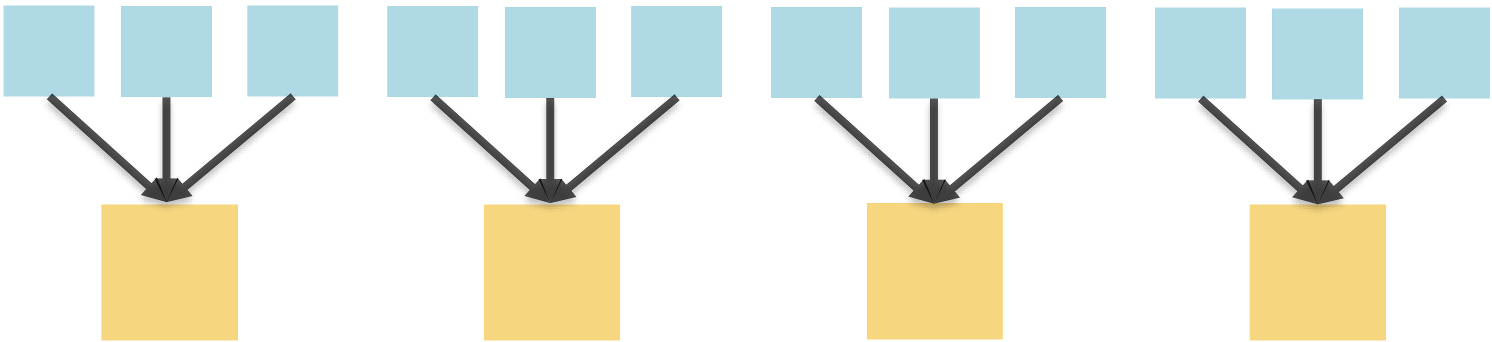


  - Given initial state + forward plan [drive-to-shop]:
    - *One* possible next state
  - Given goal states + backward plan [drive-to-shop]:
    - *Many* possible previous states

Backward Search:
Many complications – same solution

Goal States and Goals

- Main challenge: A **set** of possible ”**current**” states
  - Can’t store and process each state separately

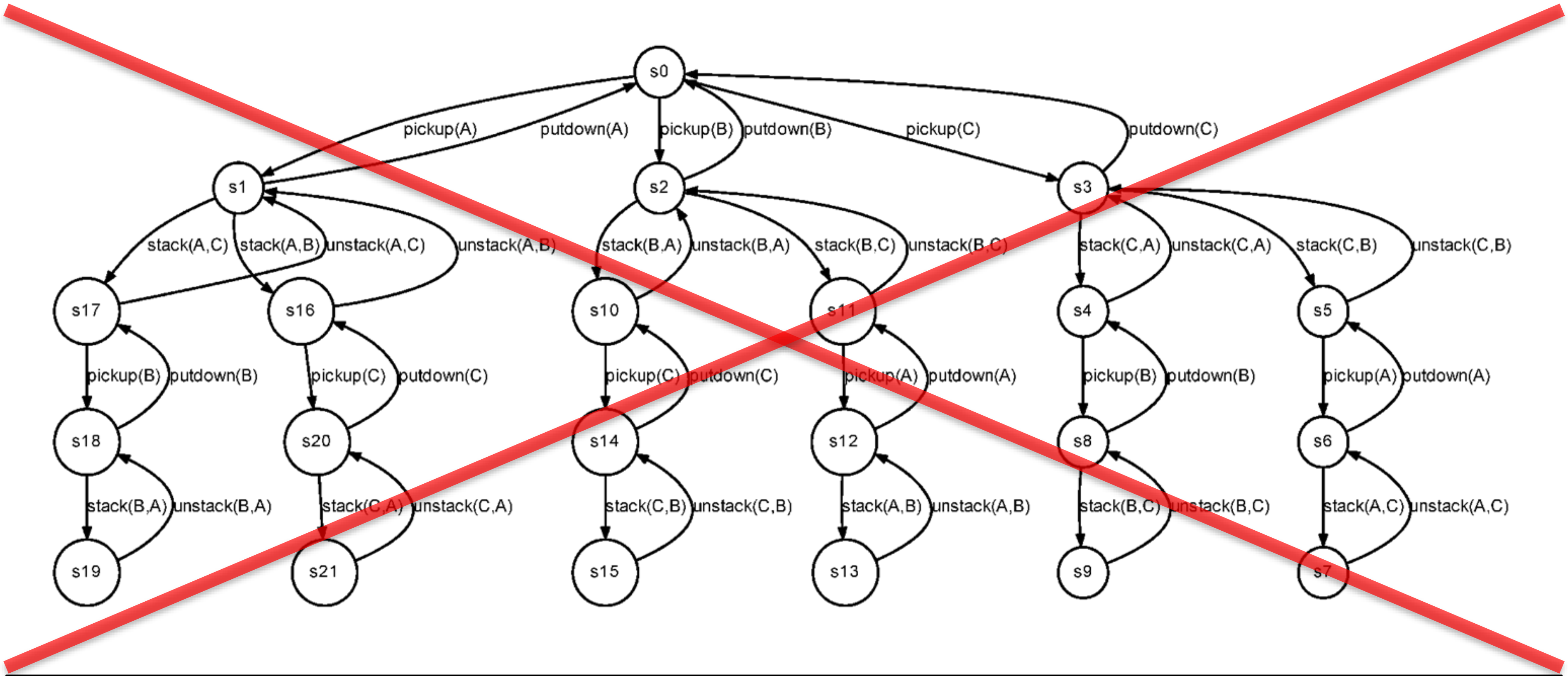


- Classical **representation**:
  - Goal: set of **literals** that **should hold**, representing multiple states
    - **g = { on(A,B), ¬on(C,D) }**
      - A should be on B, and C should *not* be on D
      - We don’t care if blocks are clear / ontable or not:
        If we cared, that would have been specified

Perfect starting point!

Goal Space ≠ State Space

Backward search uses goal space!



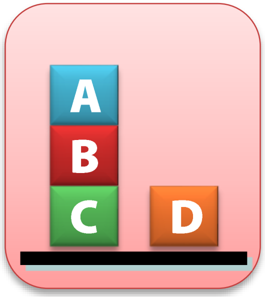
Will <u>not</u> construct <u>this</u> graph – use $\gamma^{-1}(g,a)$, not $\gamma^{-1}(s,a)$

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

Let’s see how we can construct a goal space beginning with an ”initial goal”!

Goal Specifications

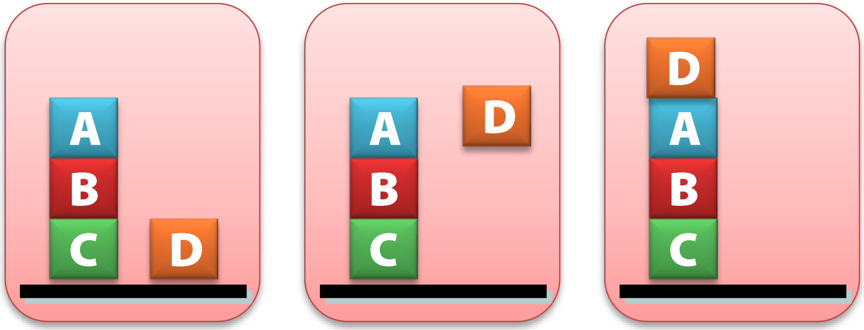
- Suppose we want exactly this:



- What is the actual **goal specification**?
  - We could specify a **complete** goal (➔ 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 might just specify this:
    - g = { on(A,B), on(B,C), ontable(C), ontable(D) }
    - Specifies all positions;
      given a *physically achievable initial state*, other facts follow implicitly

Goal Specifications (2)

- Usually we **<u>don't care</u>** about **<u>all</u>** facts (directly or indirectly)!
  - Ignore the location of block D
  - on(A,B)
    ¬clear(B)
    on(B,C)
    ontable(C)



Forward planning: ***<u>Applicability</u>***

Which actions could we ***execute****?*

Backward planning: ***<u>Relevance</u>***

Which actions could ***achieve*** *part of the goal?*

Backward Search: Relevance

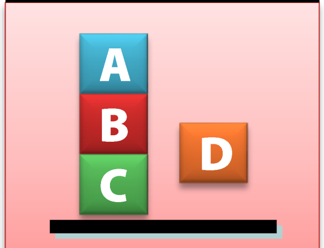
(Later:
Where would we have to start?)

What action a could be the <u>last</u> before achieving the 4 goal facts?

g: Suppose we want to achieve this…

on(A,B)
¬clear(B)
on(B,C)
ontable(C)

g specifies *some* of the facts we illustrate below…

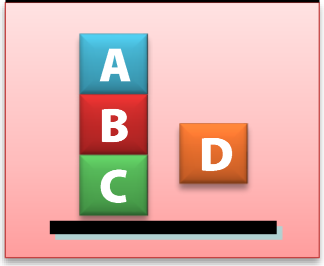


Backward Search: Relevance (2)

g:
Suppose we want to achieve this…

on(A,B)
¬clear(B)
on(B,C)
ontable(C)

g specifies *some* of the facts we illustrate below…



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

```
(: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)))
```

No!

It achieves clear(?top) = clear(B)
The goal requires ¬clear(B)

➔ Destroys part of the goal

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

Backward Search: Relevance (2)

g: Suppose we want to achieve this…

on(A,B)
¬clear(B)
on(B,C)
ontable(C)

Could **pickup(D)** be the last action in a plan achieving g?

```
(: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)))
```

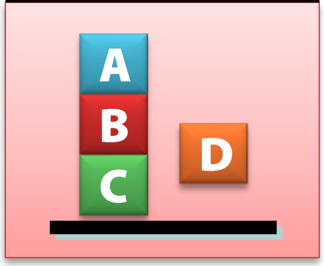
Yes! Effects:
¬ontable(D)
¬clear(D)
¬handempty
holding(D)

Does not contradict the goal

…but also doesn't help us *achieve* any goal requirements!

pickup(D) is ***not relevant***

g specifies *some* of the facts we illustrate below…



Backward Search: Relevance (3)

Yes! Effects:
¬holding(A)
¬clear(B)
clear(A)
handempty
on(A,B)

Does **not contradict** the goal, **achieves** on(A,B)

stack(A,B) is ***relevant***

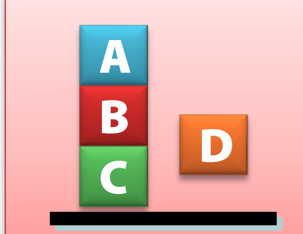
Could **stack(A,B)** be the last action in a plan achieving *g*?

```
(: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)))
```

g: Suppose we want to achieve this…

on(A,B)
¬clear(B)
on(B,C)
ontable(C)

g specifies *some* of the facts we illustrate below…



Backward Search: Summary (so far)

<u>Forward</u> search, over **<u>states</u>** $s = \{atom_1, \ldots, atom_n\}$:

a is **<u>applicable</u>** to *current state* s **iff**

precond$^+(a) \subseteq s$ and
$s \cap$ precond$^-(a) = \emptyset$

<u>Positive conditions</u> are present

<u>Negative conditions</u> are absent

<u>Backward</u> search, over **<u>sets of literals</u>** $g = \{lit_1, \ldots, lit_n\}$

a is **<u>relevant</u>** for *current goal* g **iff**

g ∩ effects(a) ≠ ∅ and
g+ ∩ effects–(a) = ∅ and
g– ∩ effects+(a) = ∅

<u>Contribute</u> to the goal
(add needed positive or negative literal)

Do not **<u>destroy</u>** any goal literals

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**?*

Progression and Regression

Forward search, over **states** $s = \{atom_1, \dots, atom_n\}$:

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

I am in state s $\xrightarrow{\text{Action } a \text{ is applicable}}$ I would end up in $\gamma(s,a)$

Backward search, over **sets of literals** $g = \{lit_1, \dots, lit_n\}$

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

I would require $\gamma^{-1}(g,a)$ $\xrightarrow{\text{Action } a \text{ is relevant for } g}$ I need to achieve goal g

Backward Search: Regression

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

What facts g'
would we require
before executing a,
so that for <u>every</u> state s
satisfying g':

1) A is **executable** in s
2) $\text{g} \subseteq \gamma(s, a)$?

*What action a
could be the <u>last</u>
before achieving
the 4 goal facts?*

g:
We want
to achieve
this…

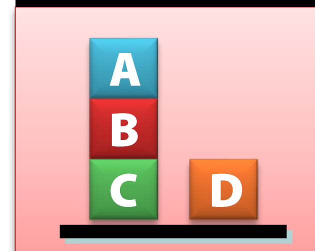
on(A,B)
on(B,C)
ontable(C)
ontable(D)

<u>Subset: It is OK to achieve more than required!</u>

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

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

g specifies *some*
of the facts we
illustrate below…



Backward Search: Regression (2)

$\gamma^{-1}(g, \textbf{stack(A,B)})$

Needed by **stack(A,B)**	holding(A) clear(B)
What the goal needs, but **stack(A,B)** did not achieve	on(B,C) ontable(C) ontable(D)

stack(A,B)

(: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)))

g:
We want to achieve this…

on(A,B)
on(B,C)
ontable(C)
ontable(D)

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

$\gamma^{-1}(g, \textbf{stack(A,B)})$ =
{ holding(A), clear(B), on(B,C), ontable(C), ontable(D) }

Corresponds to *many potential states*

g specifies *some* of the facts we illustrate below…

Backward Search: Regression (3)

- Formally:

All goals except effects(a) must already have been true

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

$\gamma^{-1}(g,a) = ((g - \text{effects}(a)) \cup \text{precond}(a))$,
representing
$\{ s \mid a \text{ is applicable to } s \text{ and } \gamma(s,a) \text{ satisfies } g \}$

Backward / regression: Which states could I start from?

Works for:

Classical goals (already sets of ground literals)
Classical effects (conjunction of literals)
Classical preconditions (conjunction of literals)

Backward Search: Keep Regressing

$g_2 = \gamma^{-1}(g, \text{stack(A,B)})$

holding(A)
clear(B)

on(B,C)
ontable(C)
ontable(D)

g_3

g_4

g_5

stack(A,B)

g:
We want to achieve this…

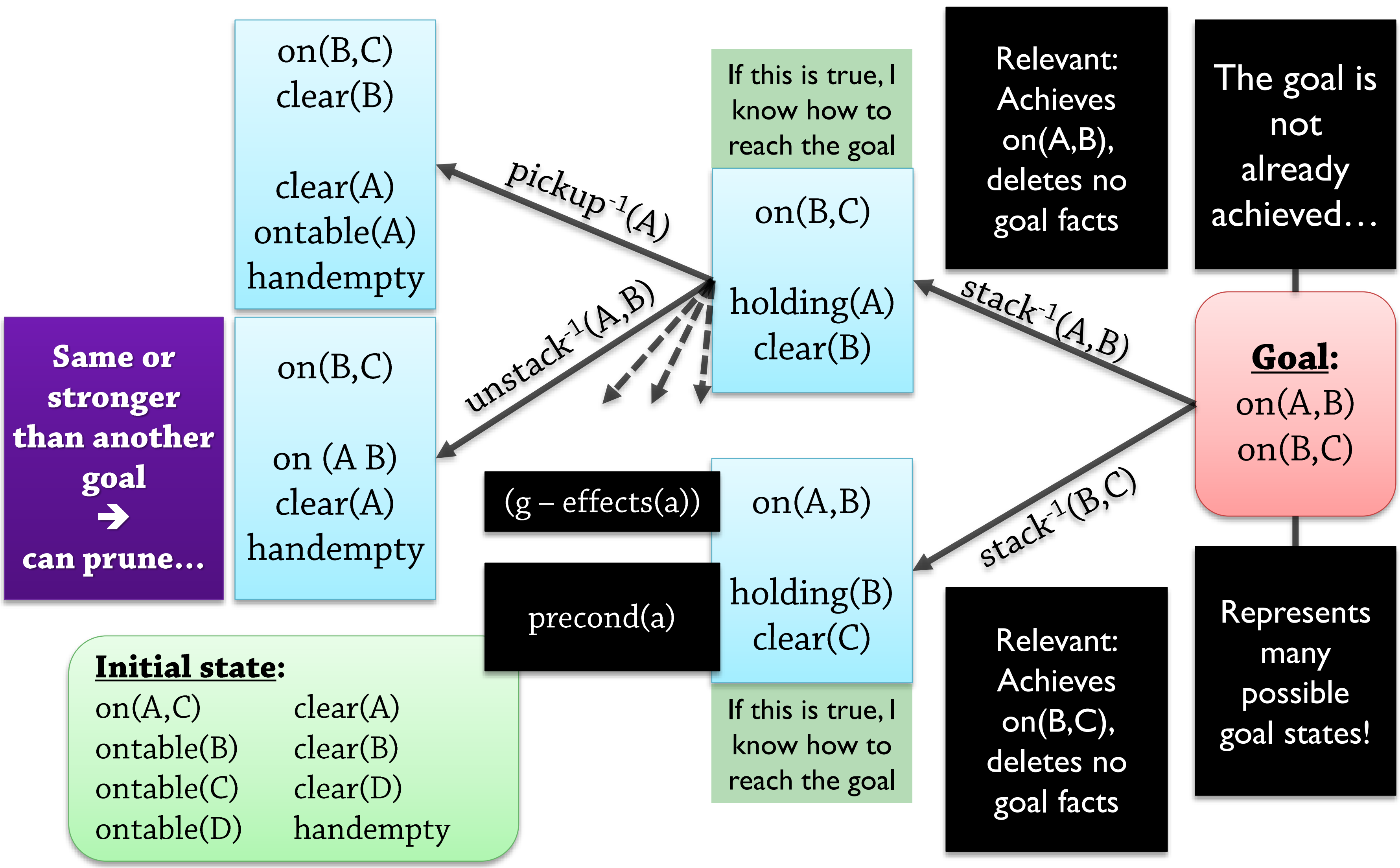
on(A,B)
on(B,C)
ontable(C)
ontable(D)

I can reach the goal from *any* state satisfying some g_i!

I can reach the goal from *any* state satisfying these 5 literals!

<u>Solution test:</u>
If the literals are satisfied in s_0,
I have a solution!

Backward Search: Example

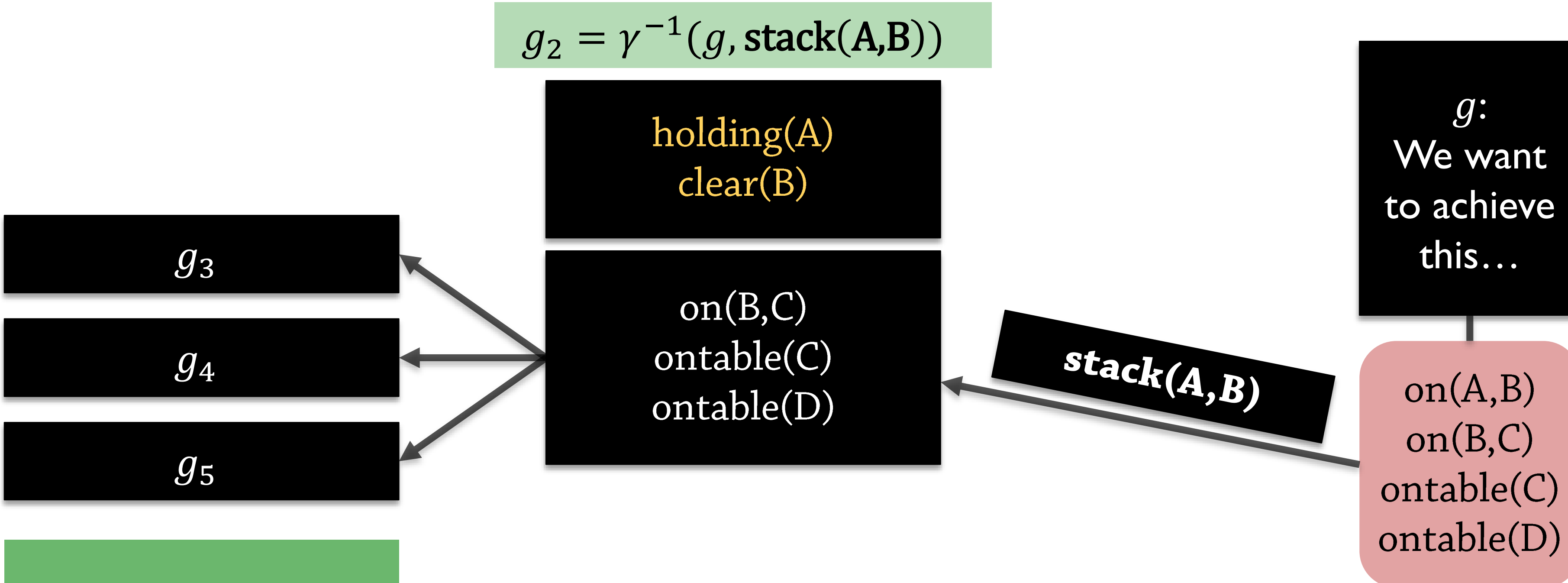


<u>When we do select actions:</u>

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



Which open node should be selected?

As usual, we need guidance!

(For example, heuristic functions)

Backward Search: New Goal Achievable?

$g_2 = \gamma^{-1}(g, \textbf{unstack(D,D)})$

on(D, D)
clear(D)
handempty

on(A,B)
on(B,C)
ontable(C)

g:
We want to achieve this…

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

unstack(D,D)

Perfectly valid action instance!

I can reach the goal from *any* state satisfying these 6 literals!

Do such states exist?

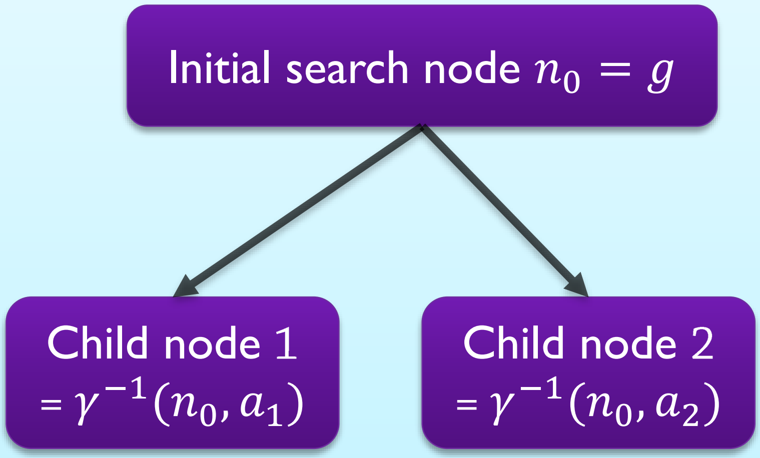
Depends on the initial state! Maybe this is exactly the initial state we specified…

If not reachable:
Can sometimes be detected automatically
➔ pruning

Backward Goal Search Space

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 *all* states s 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*

Repetition: Planning as Search


```
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

Instantiated Algorithm

- **backward-search**(A, s_0, goal) {
  initial-node ← ⟨goal, ϵ⟩ // [2]
  open ← { *initial-node* }
  while (*open* ≠ ∅) {
  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]

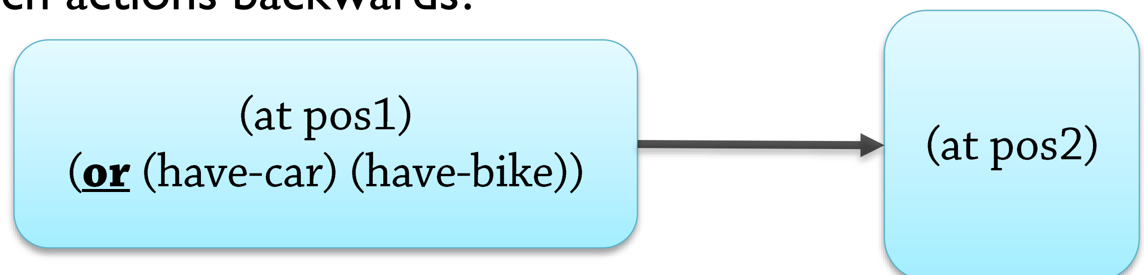
Expand node

foreach $a \in A$ relevant to g { // [3]
$g' \leftarrow \gamma^{-1}(g, a)$
$\pi' \leftarrow$ **append**(a, π)
add ⟨g', π'⟩ to *open*
}
}
// Expanded the entire search space without finding a solution
return failure;
}

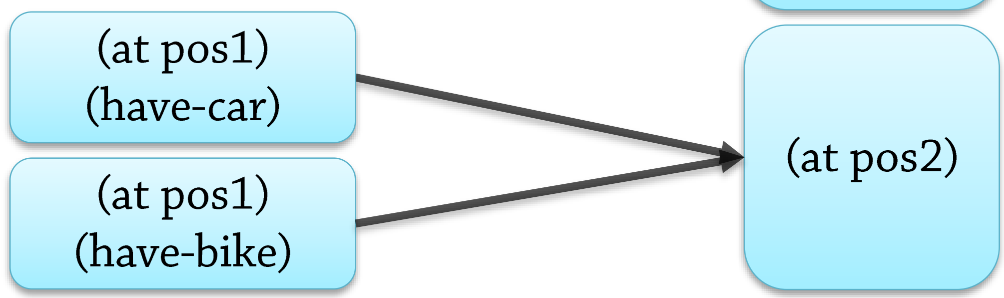
Expressivity Constraints

Backward and Forward Search: Expressivity

- How about **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)))
      :**effects** (**and** (at ?to) (not (at ?from))))
  - How do we apply such actions backwards?
    - More complicated **disjunctive goals** to achieve?



    - Additional **branching**?

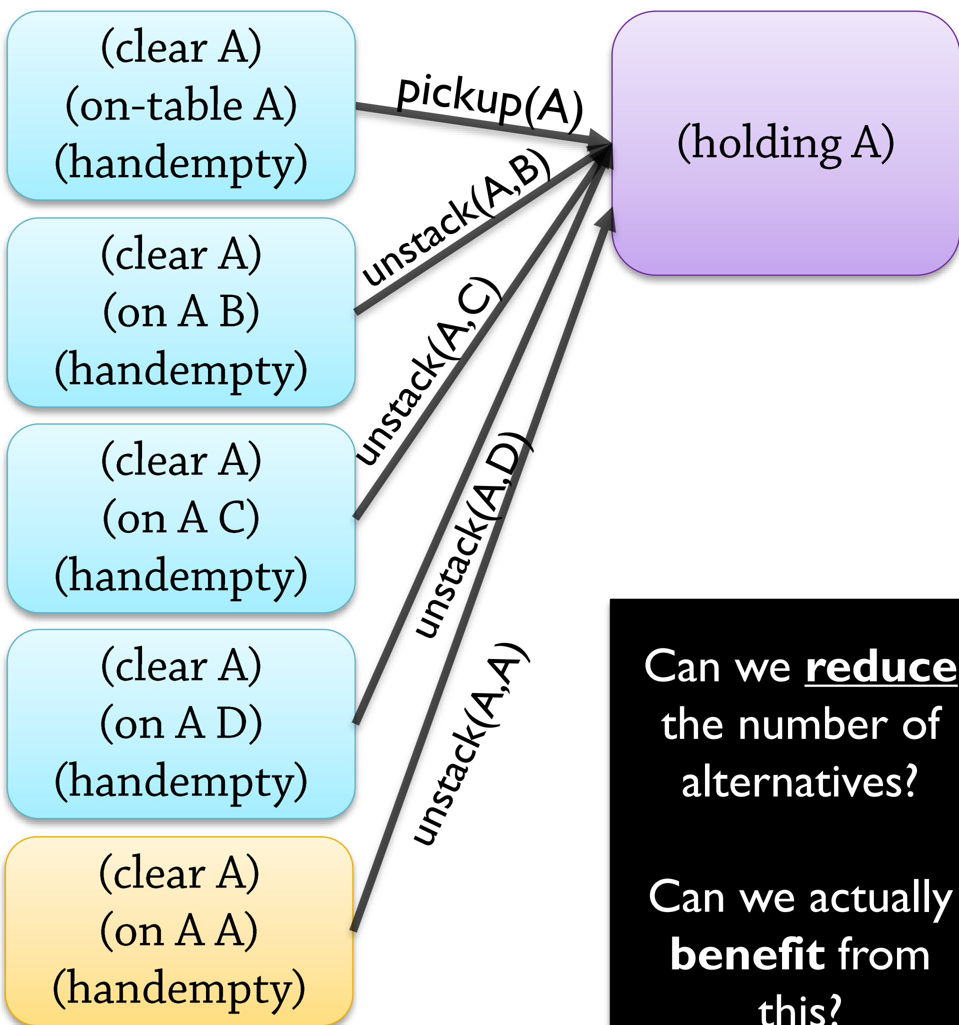


Similarly for existentials ("**exists** block [on(block,A)]"): One branch per possible value
Some extensions are less straight-forward in backward search (but possible!)

Lifted Search:
A general technique

Lifted Search 1: Motivation

- Potential problem in any search space: **high branching factors**



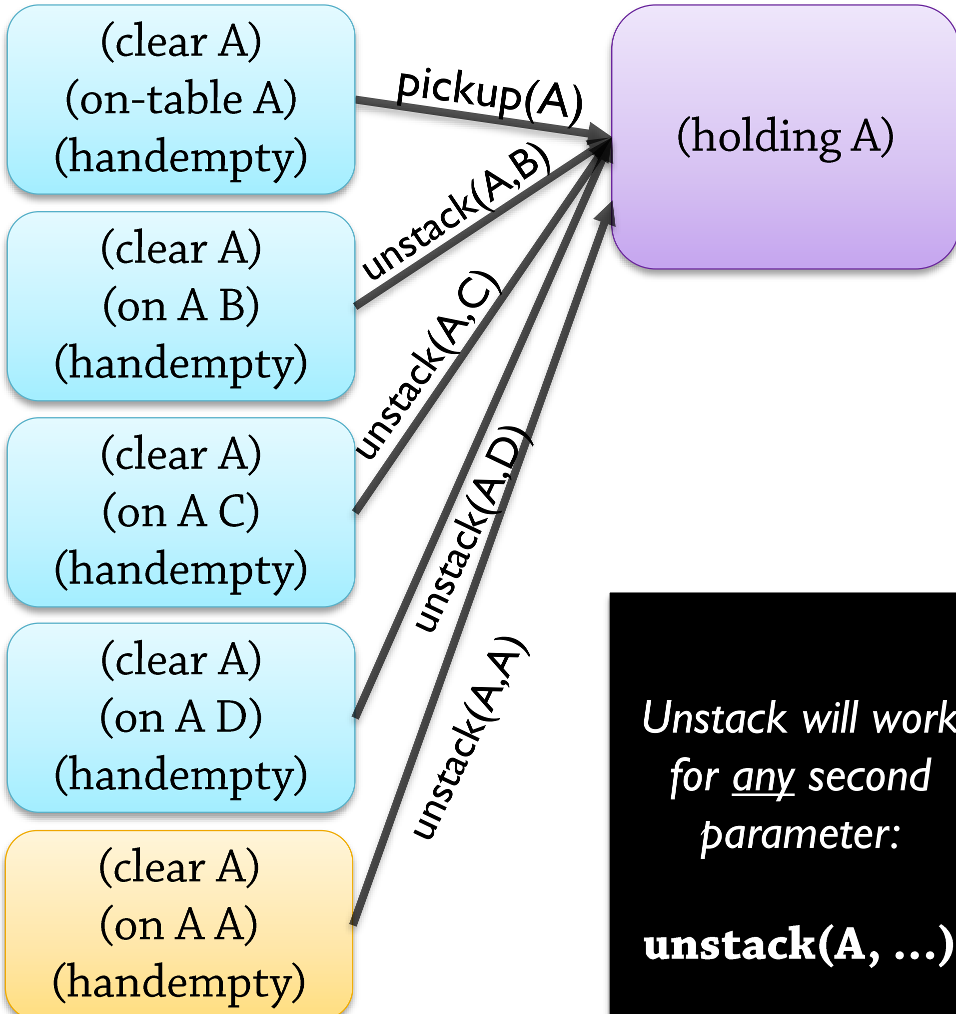

```
(:action pickup
  :parameters (?x)
  :precondition (and (clear ?x) (on-table ?x)
                 (handempty))
  :effect
  (and (not (on-table ?x))
       (not (clear ?x))
       (not (handempty))
       (holding ?x)))

(: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))))
```

Can we **reduce** the number of alternatives?

Can we actually **benefit** from this?

Lifted Search 2: Observation



Unstack will work for any second parameter:

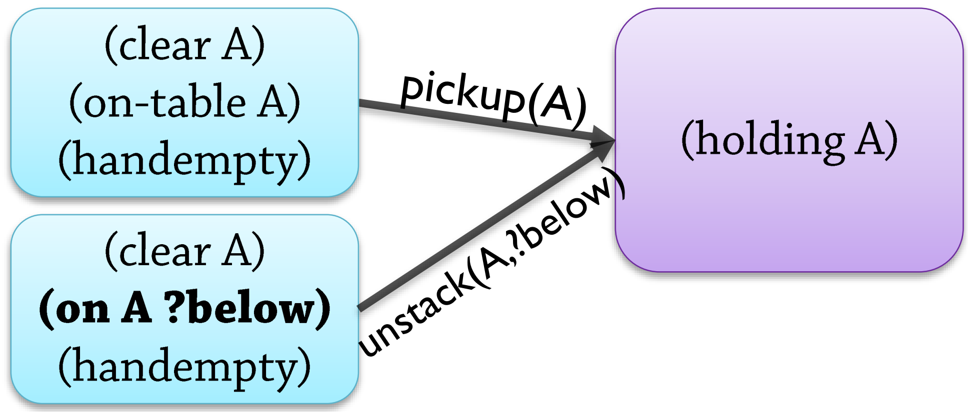
unstack(A, ...)

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

Lifted Search 3: General Idea

- General idea in *lifted search*:
  - Instantiate parameters that are "bound" by the goal (as usual)
    - For (pickup ?x) to achieve (holding A), we *must* have ?x == A
  - Keep other parameters **uninstantiated**
    - For (unstack ?top ?below) to achieve (holding A), we *must* have ?top == A
    - We don't care about ?below, so don't give it a value: use **unstack(A, ?below)**
  - Not *ground* ➔ "lifted"!



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

Suppose (on A B) is true initially, *or* made true by action A1

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

Only **two** new nodes to keep track of!

Applicable to other types of planning – will return later!