Planning and Scheduling: State-Space Planning

Hochschule Bonn-Rhein-Sieg

b-it Bonn-Aachen International Center for Information Technology

Prof. Dr.-Ing. Gerhard K. Kraetzschmar

Acknowledgements

- These slides are based on those slides by Dana Nau, and Reid Simmons
- Some improvements have been implemented by Iman Awaad

Motivation

- Planning as search…
- Which search space?

- State Space
  - Each node represents a state of the world
  - A plan is a path through the space

- Plan Space
  - Each state is a set of partially instantiated operators and some constraints
  - Impose more and more constraints until we get a plan

Linear Search

- Basic Idea:
  - Work on one goal until completely solved before moving on to the next goal
  - Order in which problems are solved is linearly-related to the order in which the plan actions are executed
- Planning algorithms maintain a goal stack
- Implications:
  - No interleaving of goal achievement
  - Efficient search if goals do not interact (much)
- Search space is still larger than it should be…

Means-End Analysis

- Basic Idea:
  - Search only relevant aspects of problem
  - What means (operators) are available to achieve the desired ends (goals)

1) Find difference between goal and current state
2) Find operator to reduce difference
3) Perform means-end analysis on new sub-goals…

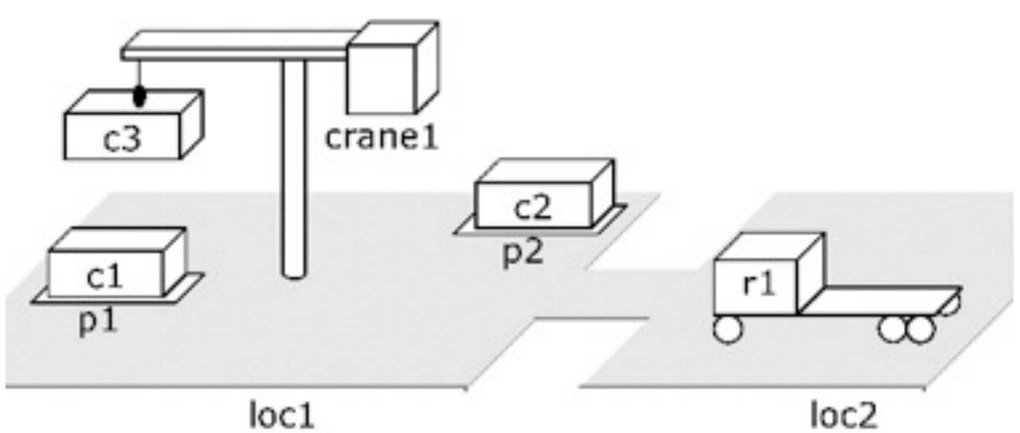
- Introduced by Newell, Simon, Ernest: General Problem Solver (GPS) [in the 1960s]

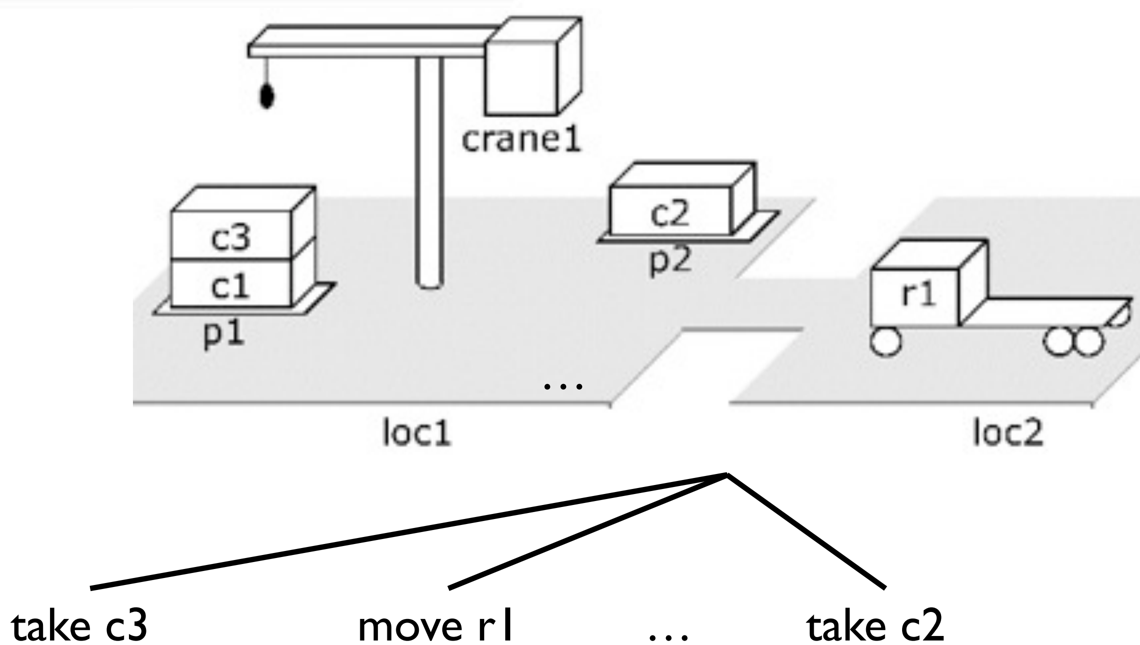
Outline

- Forward Search
- Backward Search
- Lifting
- STRIPS (Fikes, Nielson 1971)
  - Same idea as GPS,
  - but solved the frame problem with the STRIPS assumption,
  - introduced operator representation,
  - operationalized ideas of difference, sub-goals and applicability
  - dealt (to some degree) with plan execution and learning
- Block stacking

Forward Search

Forward-search(O, s_0, g)
$s \leftarrow s_0$
$\pi \leftarrow$ the empty plan
loop
if s satisfies g then return π
$E \leftarrow \{a | a$ is a ground instance an operator in O,
and $\mathrm{precond}(a)$ is true in $s\}$
if $E = \emptyset$ then return failure
nondeterministically choose an action $a \in E$
$s \leftarrow \gamma(s, a)$
$\pi \leftarrow \pi.a$



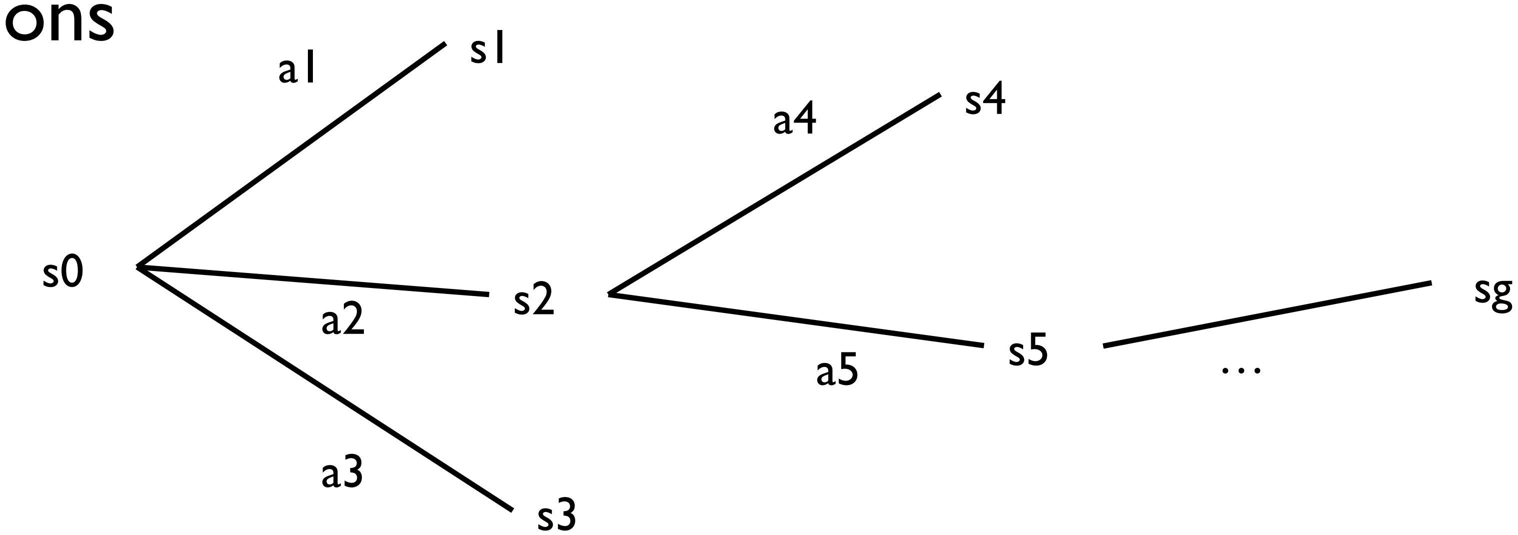


Properties

- Forward-search is sound:
  - For any plan returned by any of its *nondeterministic* traces, this plan is guaranteed to be a solution.
- Forward-search also is complete:
  - If a solution exists,
    then at least one of Forward-search's *nondeterministic* traces will return a solution.

Deterministic Implementations

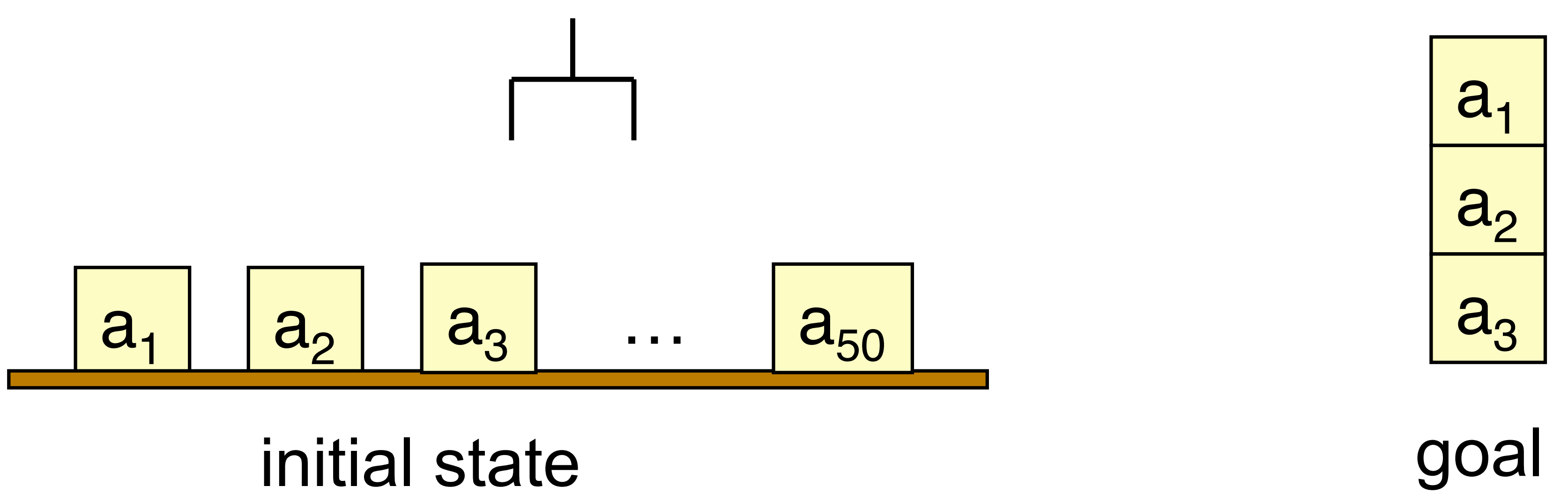
- Some *deterministic* implementations of forward search:
  - breadth-first search
  - best-first search
  - depth-first search
  - greedy search



- Breadth-first and best-first search are *sound* and *complete*
  - But they usually aren't practical because they require too much memory
  - Memory requirement is exponential in the length of the solution
- In practice, more likely to use a depth-first search or greedy search
  - Worst-case memory requirement is linear in the length of the solution
  - Sound but not complete
    - But classical planning has only finitely many states
    - Thus, can make depth-first search complete by doing loop-checking

Branching Factor of Forward Search

- Forward search can have a very large branching factor (see example)
- Why this is bad:
  - Deterministic implementations can waste time trying lots of irrelevant actions
- Need a good heuristic function and/or pruning procedure
  - See Section 4.5 (Domain-Specific State-Space Planning) and Part III (Heuristics and Control Strategies)



- Search space is still larger than it should be…

Backward Search

- Use means-end-analysis: search only *relevant* aspects of the problem
- For forward search, we started at the initial state and computed state transitions:
  - new state $s' = \gamma(s, a)$
- For backward search, we start at the goal and compute inverse state transitions
  - new set of subgoals $g' = \gamma^{-1}(g, a)$

Inverse State Transitions

- What do we mean by $\gamma^{-1}(g,a)$?
- First need to define relevance:
  - An action a is relevant for a goal g if
    - a makes at least one of g's literals true

      $$g \cap effects(a) \neq \varnothing$$

    - a does not make any of g's literals false

      $$g_+ \cap effects_-(a) = \varnothing$$
      $$g_- \cap effects_+(a) = \varnothing$$

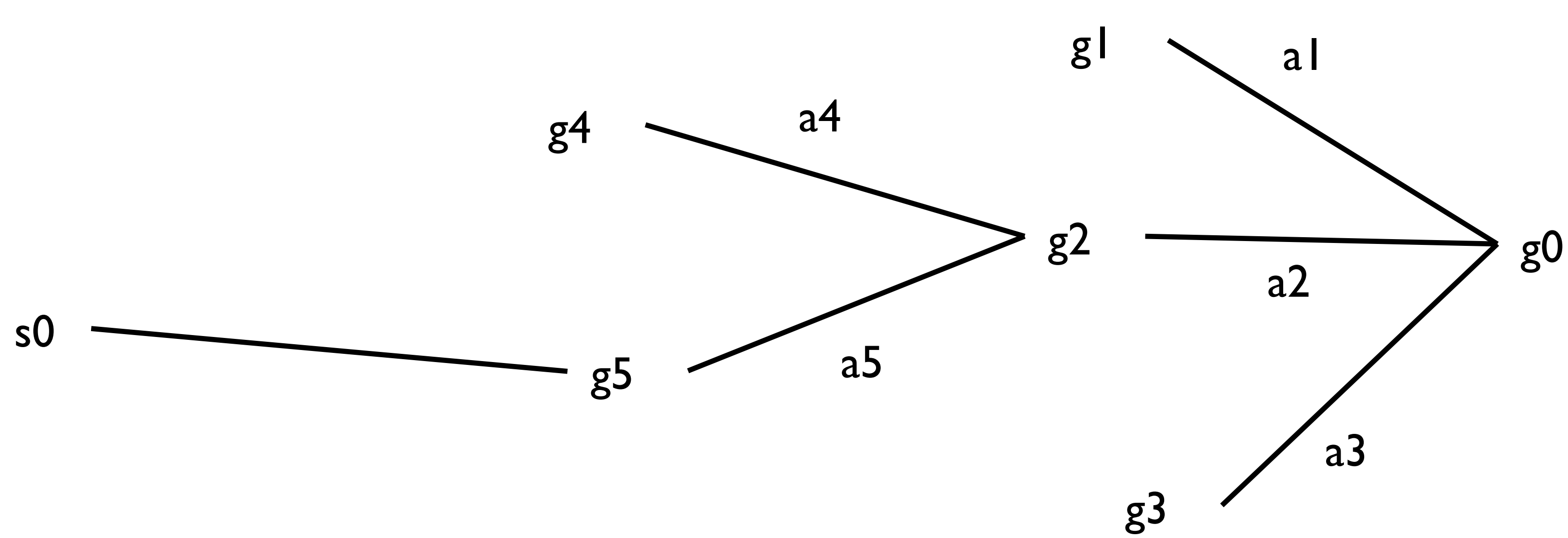
- If a is relevant for g, then

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

E.g.:
g = {on(b1,b2), on(b2,b3)}
a = stack(b1,b2)
What is $\gamma^{-1}(g,a)$?

Backward Search Algorithm

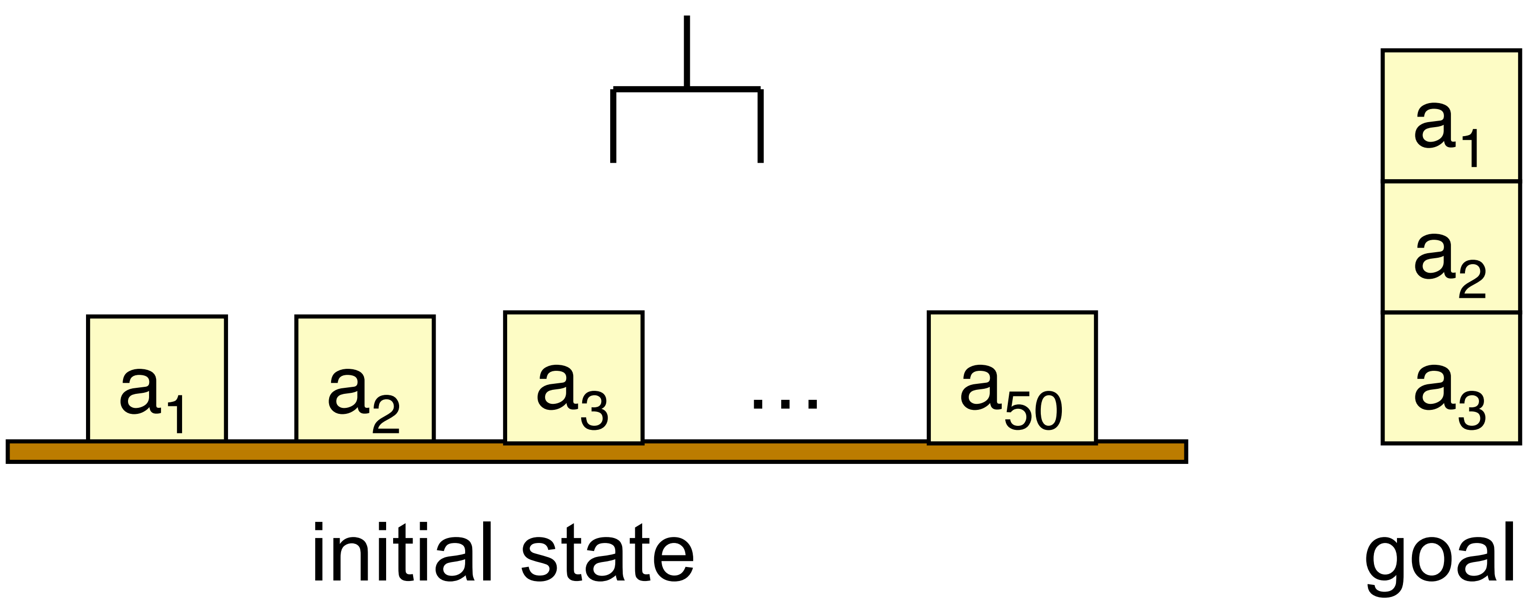
Backward-search(O, s_0, g)
  $\pi \leftarrow$ the empty plan
  loop
    if s_0 satisfies g then return π
    $A \leftarrow \{a | a$ is a ground instance of an operator in O
        and $\gamma^{-1}(g, a)$ is defined$\}$
    if $A = \emptyset$ then return failure
    nondeterministically choose an action $a \in A$
    $\pi \leftarrow a.\pi$
    $g \leftarrow \gamma^{-1}(g, a)$

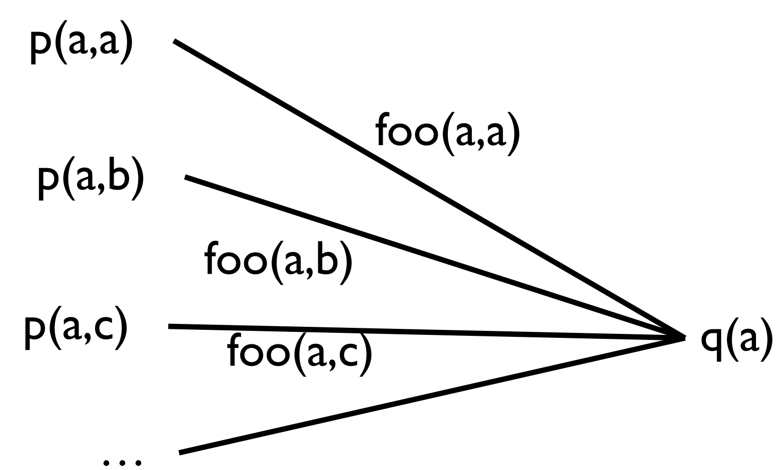


Efficiency of Backward Search

- Backward search's branching factor is small in our example
- There are cases where it can still be very large
  - Many more operator instances than needed

foo(x,y)
 precond: p(x,y)
 effects: q(x)

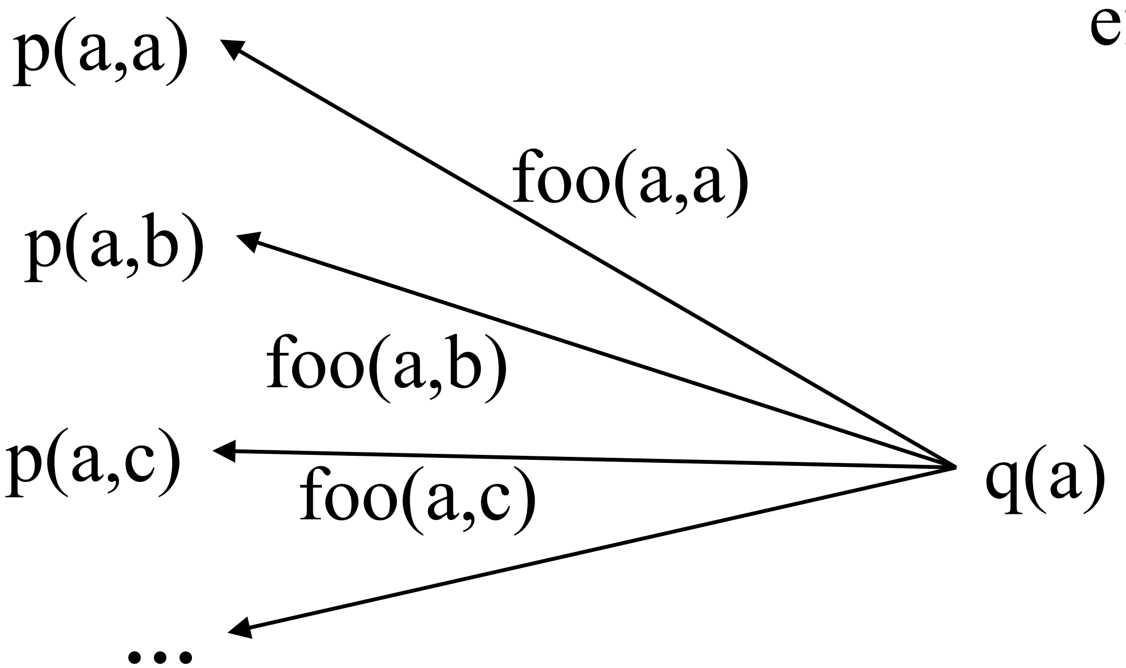


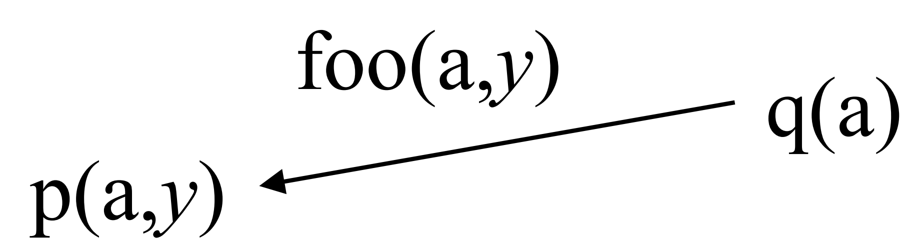


Lifting

- We can reduce the branching factor
  if we partially instantiate the operators
  - this is called lifting

foo(x,y)
  precond: p(x,y)
  effects: q(x)





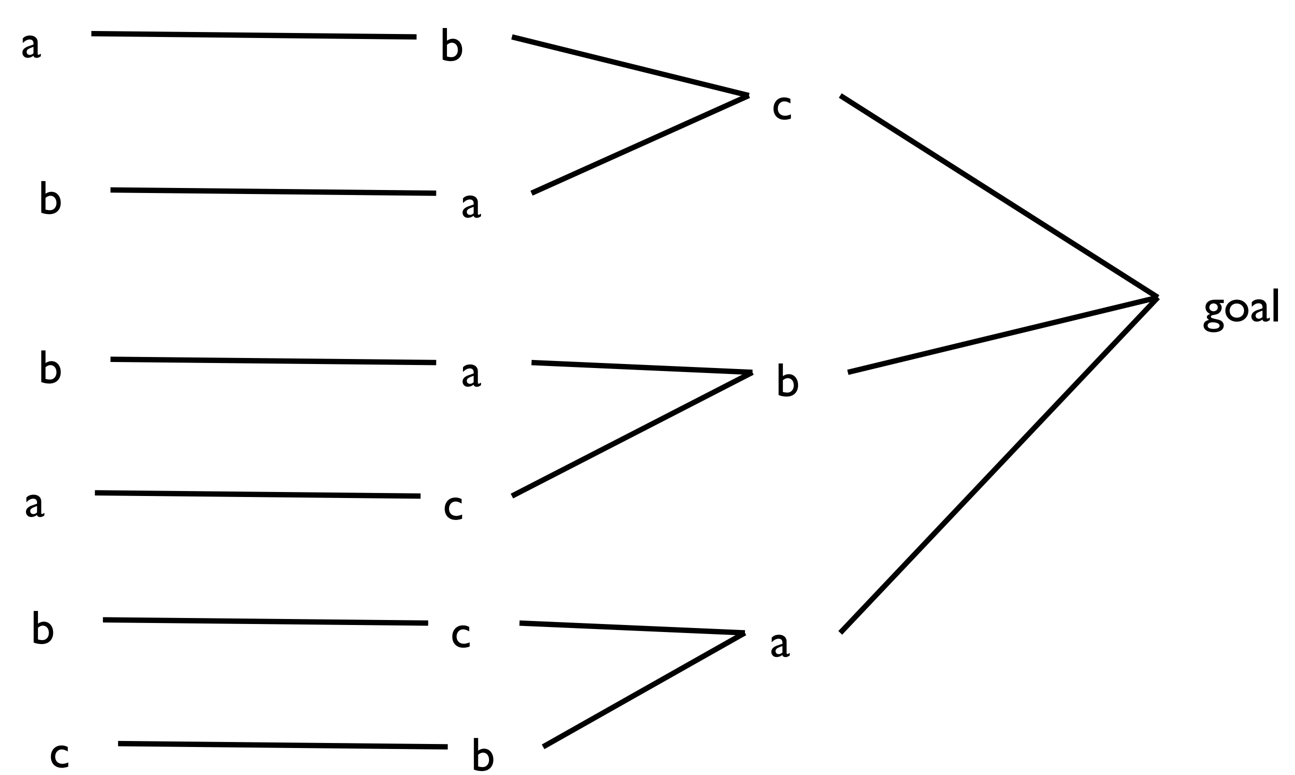
Lifted Backward Search

```
Lifted-backward-search(O, s_0, g)
    π ← the empty plan
    loop
        if s_0 satisfies g then return π
        A ← {(o, θ) | o is a standardization of an operator in O,
                  θ is an mgu for an atom of g and an atom of effects⁺(o),
                  and γ⁻¹(θ(g), θ(o)) is defined}
        if A = ∅ then return failure
        nondeterministically choose a pair (o, θ) ∈ A
        π ← the concatenation of θ(o) and θ(π)
        g ← γ⁻¹(θ(g), θ(o))
```

- More complicated than Backward-search
  - Have to keep track of what substitutions were performed
- But it has a much smaller branching factor

The Search Space is Still Too Large

- Lifted-backward-search generates a smaller search space than Backward-search, but it still can be quite large
  - If some sub-problems are independent and something else causes problems elsewhere, we'll try all possible orderings before realizing there is no solution
  - More about this in Chapter 5 (Plan-Space Planning)



Other Ways to Reduce the Search

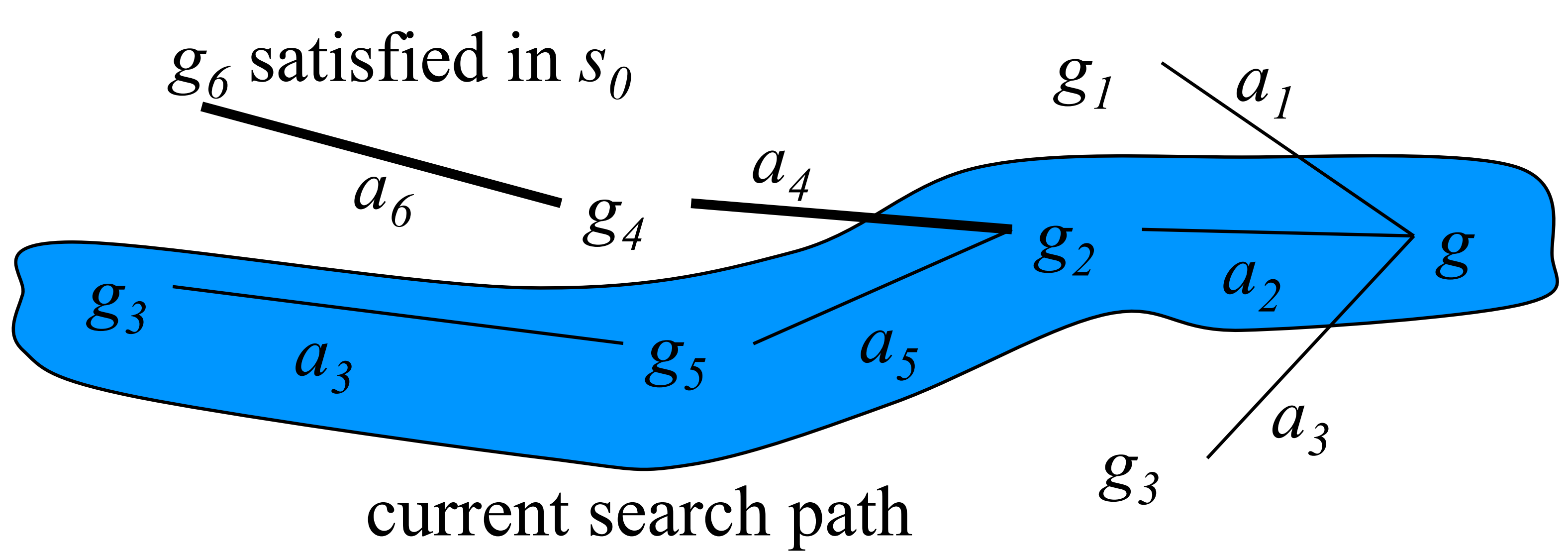
- Search Control Strategies
  - Part III of the textbook - E.g.: Least Commitment Strategies
- For now - two examples:
  - STRIPS
  - Block stacking

STRIPS

- $\pi \leftarrow$ the empty plan
- do a modified backward search from g
  - instead of $\gamma^{-1}(s, a)$, each new set of sub-goals is just $precond(a)$
  - whenever you find an action that's executable in the current state, then go forward on the current search path as far as possible, executing actions and appending them to π
  - repeat until all goals are satisfied

$$\pi = \langle a_6, a_4 \rangle$$

$$s = \gamma(\gamma(s_0, a_6), a_4)$$



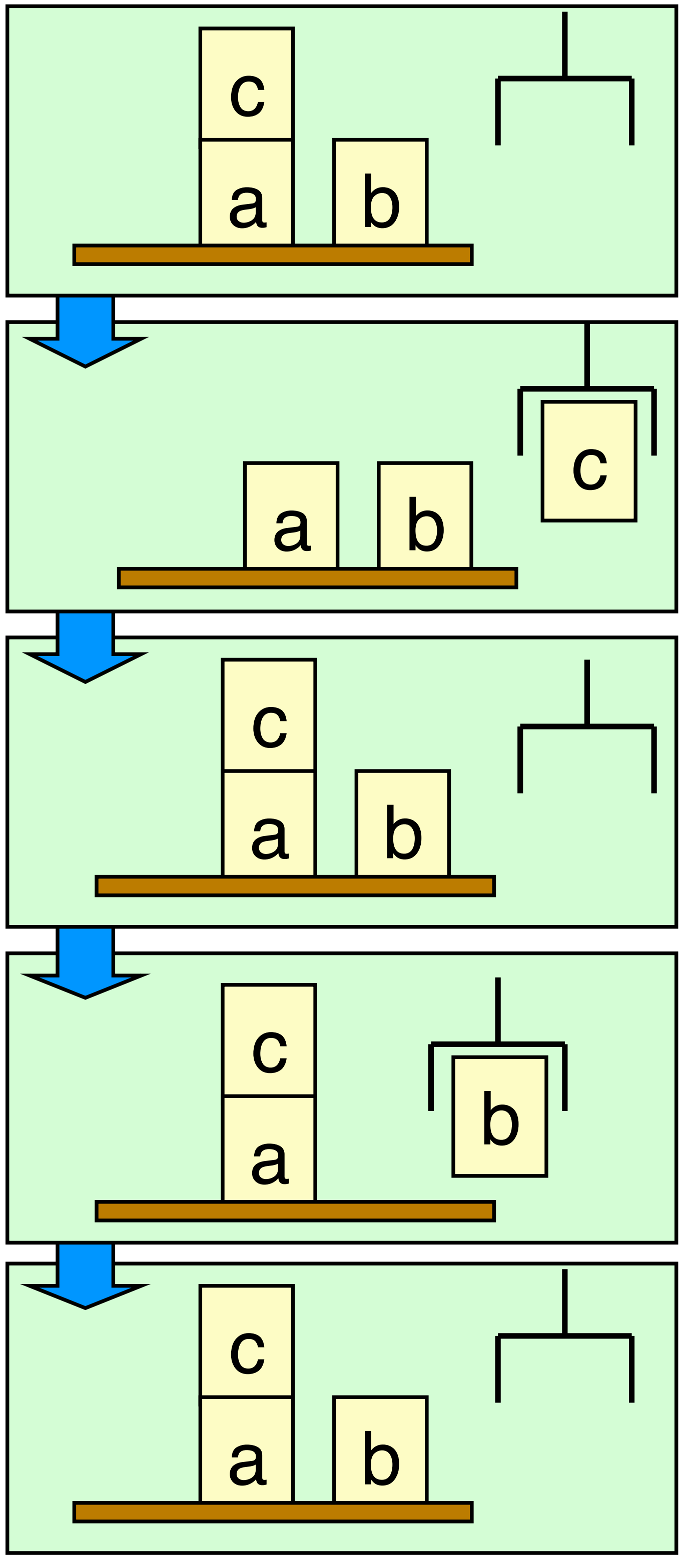
Quick Review of Blocks World

unstack(x,y)
Pre: on(x,y), clear(x), handempty
Eff: ~on(x,y), ~clear(x), ~handempty,
holding(x), clear(y)

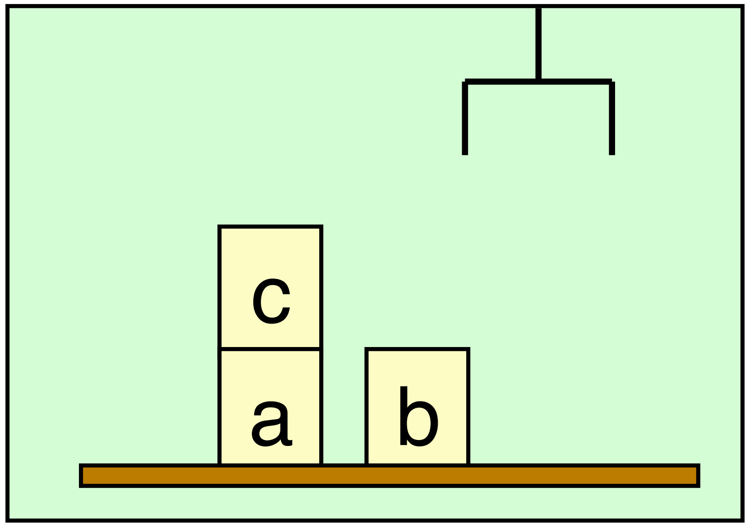
stack(x,y)
Pre: holding(x), clear(y)
Eff: ~holding(x), ~clear(y),
on(x,y), clear(x), handempty

pickup(x)
Pre: ontable(x), clear(x), handempty
Eff: ~ontable(x), ~clear(x), ~handempty, holding(x)

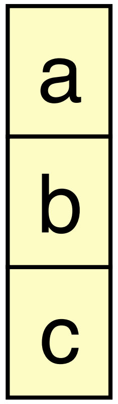
putdown(x)
Pre: holding(x)
Eff: ~holding(x), ontable(x), clear(?x), handempty



The Sussman Anomaly



Initial state



Goal

- For this problem, STRIPS can not produce an irredundant solution!
  - Try and see!
- Shows the weakness of non-interleaved planning
- Problem in the 'divide and conquer' strategy of the planner

The Register Assignment Problem

State-variable formulation:

- Initial State: {value(r1)=3, value(r2)=5, value(r3)=0}
- Goal: {value(r1)=5, value(r2)=3}
- Operator: assign(r,v,r',v')
  - Preconditions: value(r)=v, value(r')=v'
  - Effects: value(r)=v'

- STRIPS can not solve this problem at all!

Linear Planning: Discussion

- Advantages
  - Reduced search space, since goals are solved one at a time
  - Advantageous if goals are (mainly) independent
  - Linear planning is sound
- Disadvantages
  - Linear planning may produce suboptimal solutions (based on the number of operators in the plan)
  - Linear planning is incomplete

How to Fix Linear Planning?

- Several ways:
  - Do something other than state-space search
    - e.g., Chapters 5–8 in the text book
  - Use forward or backward state-space search, with domain-specific knowledge to prune the search space
    - Can solve both problems quite easily this way
    - Example: block stacking using forward search

Domain-Specific Knowledge

- A blocks-world planning problem $P = (O, s_0, g)$ is solvable if s0 and g satisfy some simple consistency conditions:
  - g should not mention any blocks not mentioned in s0
  - a block cannot be on two other blocks at once
  - etc.

Can check these in time O(n log n)

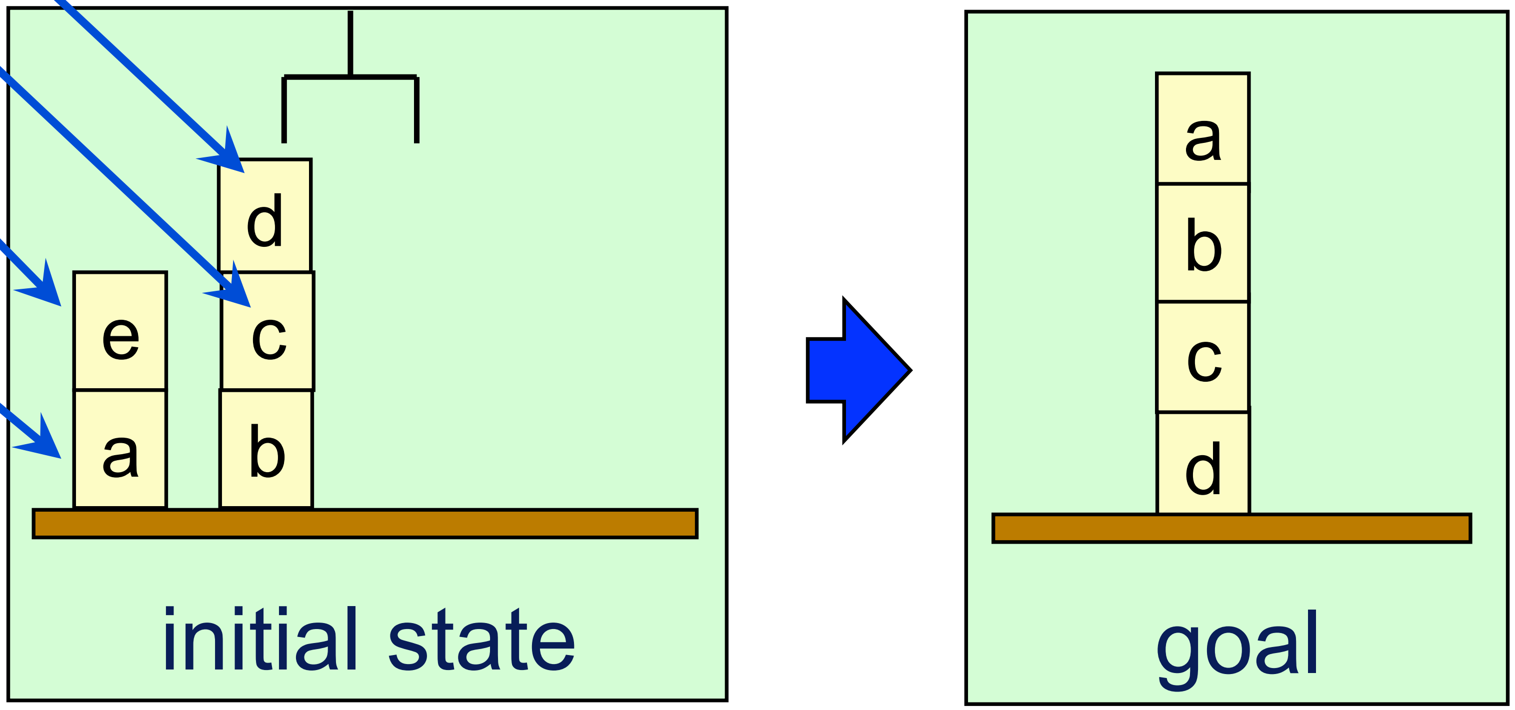
- If P is solvable, can easily construct a solution of length O(2m), where m is the number of blocks
  - Move all blocks to the table, then build up stacks from the bottom

Can do this in time O(n)

- With additional domain-specific knowledge can do even better…

Additional Domain-Specific Knowledge

- A block x needs to be moved if any of the following is true:
  - s contains ontable(x) and g contains on(x,y)
  - s contains on(x,y) and g contains ontable(x)
  - s contains on(x,y) and g contains on(x,z) for some y≠z
  - s contains on(x,y) and y needs to be moved



Domain-Specific Algorithm

loop

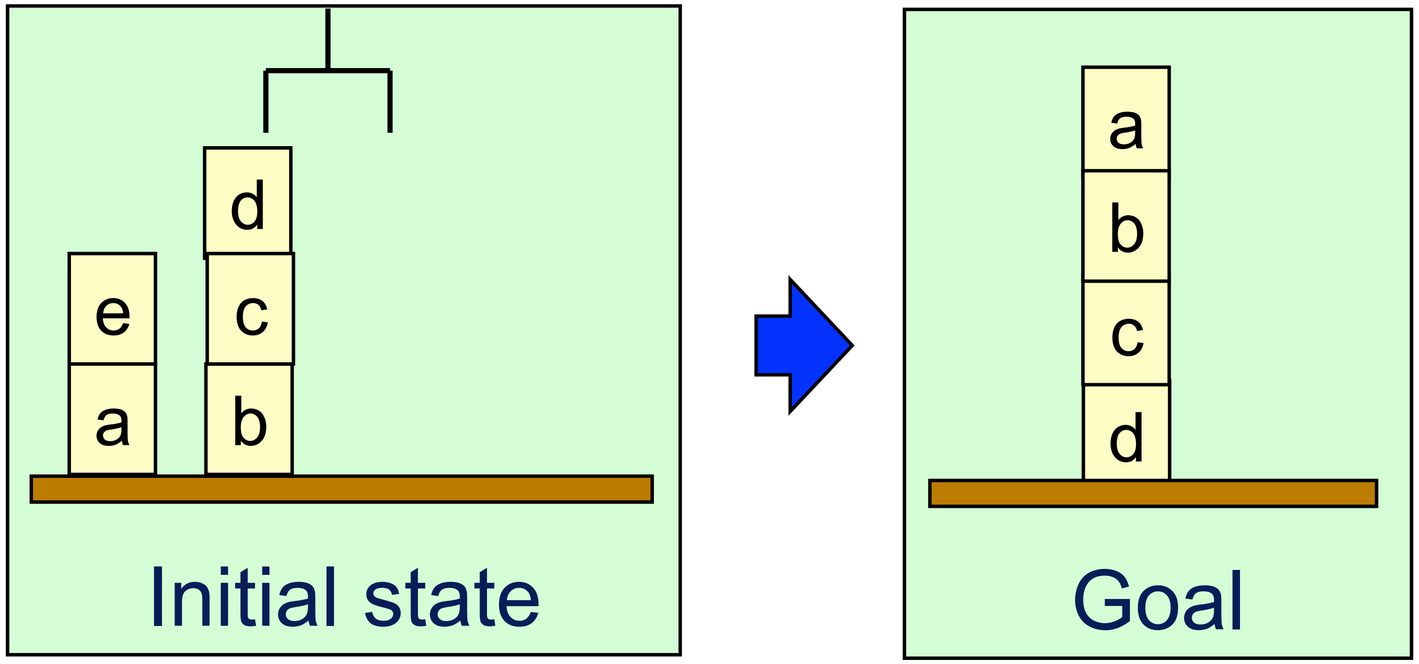
a
- **if** there is a clear block x such that
  - x needs to be moved **and**
  - x can be moved to a place where it won't need to be moved
- **then** move x to that place

b
- **else if** there is a clear block x such that
  - x needs to be moved
- **then** move x to the table

c
- **else if** the goal is satisfied
- **then return** the plan

else return failure

repeat



Easily Solves the Sussman Anomaly

loop

if there is a clear block x such that

x needs to be moved **and**

x can be moved to a place where it won't need to be moved

then move x to that place

else if there is a clear block x such that

x needs to be moved

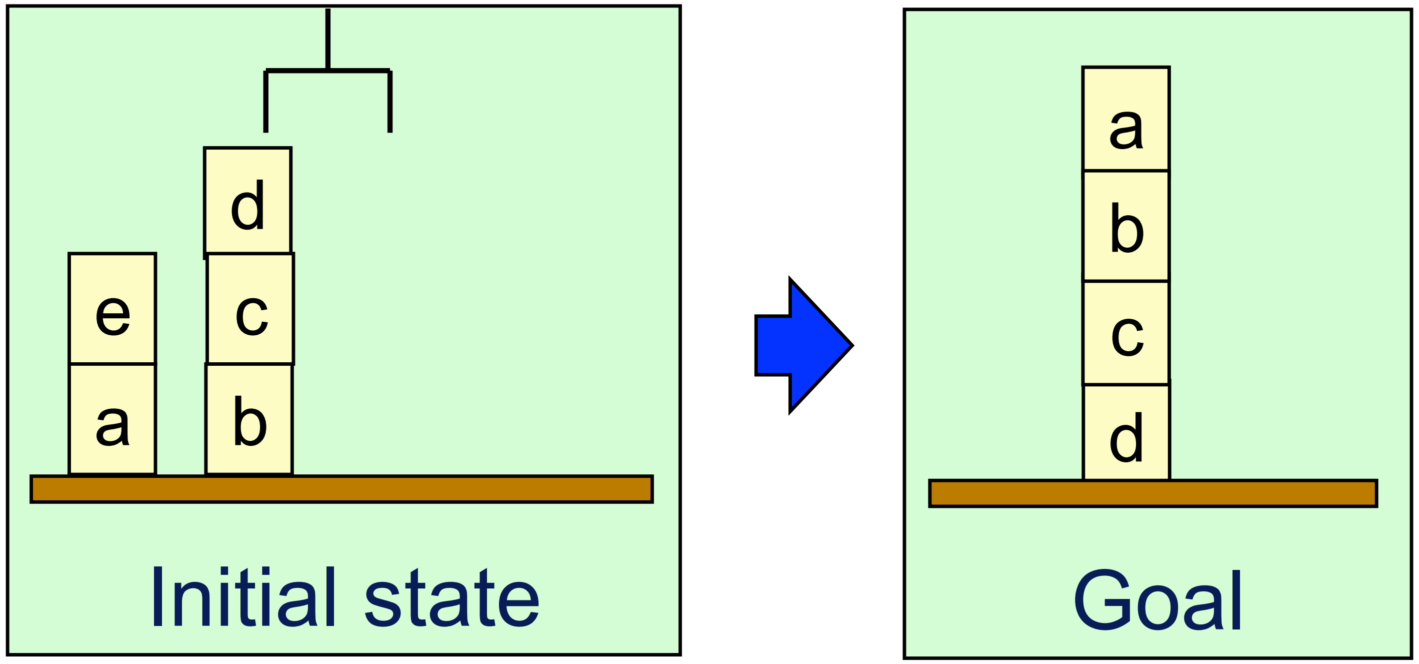
then move x to the table

else if the goal is satisfied

then return the plan

else return failure

repeat



Properties

- The block-stacking algorithm is:
  - Sound, complete, guaranteed to terminate
  - Runs in time $O(n^3)$ Can can be modified to run in time $O(n)$
  - Often finds optimal (shortest) solutions
  - But sometimes only near-optimal (Exercise 4.22 in the book)
    - Recall that PLAN-LENGTH is NP-complete

Next Week: Non-Linear Planning

- Basic Idea:
  - Use goal set instead of goal stack
  - Include in the search space all possible sub-goal orderings
    - Handles goal interactions by interleaving
- Advantages
  - Sound & Complete
  - May be optimal with respect to plan length
    (depending on search strategy employed)
- Disadvantages
  - Larger search space,
    since all possible goal orderings may have to be considered
  - Somewhat more complex algorithm; more bookkeeping