

Intelligent Agents

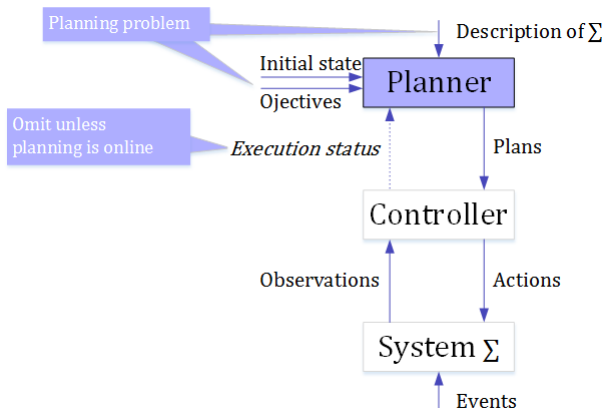
State-Space Planning

Ute Schmid

Cognitive Systems, Applied Computer Science, Bamberg University

last change: 14. April 2016

Today: The Planner



Computing A Sequence of Actions

- Previous lecture: How to transform a state into a successor state by applying an action ($\gamma(s, a) = s'$)
- Today: Compute a plan – a sequence of action applications to transform an initial state into a state fulfilling all objectives (goals)

Motivation

- Nearly all planning procedures are search procedures
- Different planning procedures have different search spaces

- Two examples:

⇒ *State-space planning*

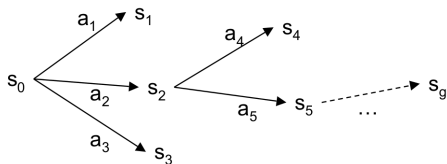
- Each node represents a state of the world
 - A plan is a path through the space

⇒ *Plan-space planning*

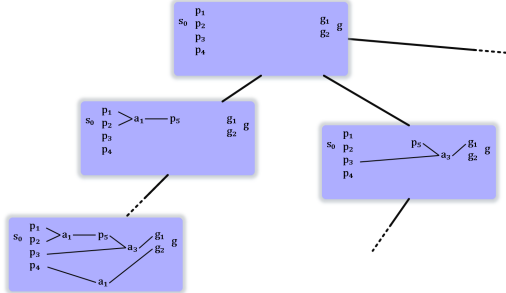
- Each node is a set of partially-instantiated operators, plus some constraints
 - Impose more and more constraints, until we get a plan

Motivation

State-Space-Planning



Plan-Space-Planning



Outlook

- Forward Search
- Backward Search
 - Inverse State Transition
 - Lifting
- Soundness, Completeness, Efficiency
- Strips
- Incompleteness of Linear Planning
 - Sussman Anomaly
- Domain Specific Knowledge

Forward-Search

Algorithm 1 Forward-search(O, s_0, g)

$s \leftarrow s_0$

$\pi \leftarrow$ the empty plan

loop

 if s satisfies g then return π

$E \leftarrow \{a \mid a \text{ is a ground instance of an operator in } O, \text{ and } \text{precond}(a) \text{ is true in } s\}$

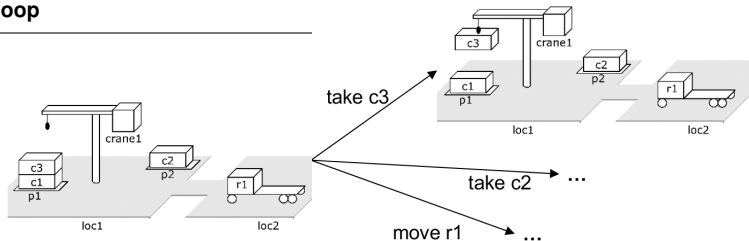
 if $E = \emptyset$ then return failure

 non-deterministically choose any action $a \in E$

$s \leftarrow \gamma(s, a)$

$\pi \leftarrow \pi.a$

end loop



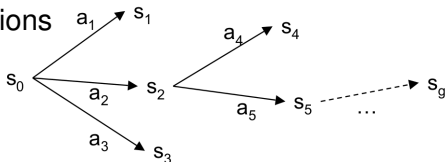
Properties

- Forward-search is *sound*
 - for any plan returned by any of its non-deterministic 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 non-deterministic traces will return a solution.
- **Remarks on non-determinism:**
 - In Algorithm 1 and further algorithms, no strategy for selecting an action is fixed.
 - Non-deterministic selection as an abstract concept guarantees that the “right” actions can be selected and in consequence that a plan can be found if one exists.
 - In practice, a deterministic action selection strategy has to be implemented. This strategy might be incomplete.

Deterministic Implementations

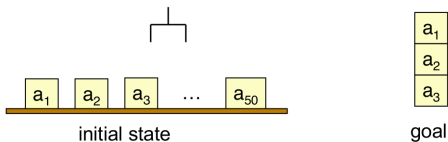
- Some deterministic implementations of forward search:

- breadth-first search
- depth-first search
- best-first search (e.g., A^*)
- 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 depth-first search or greedy search
 - Worst-case memory requirement is linear in the length of the solution
 - In general, 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
 - E.g., many applicable actions that don't progress toward goal
- 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)
 - in Ghallab, Malik, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Elsevier, 2004.
 - and lecture on Heuristic Search Planning

Backward Search

- For forward search, we started at the initial state and computed state transitions
 - new state = $\gamma(s, a)$
- For backward search, we start at the goal and compute inverse state transitions
 - new set of sub-goals = $\gamma^{-1}(g, a)$
- To define $\gamma^{-1}(g, a)$, must first define *relevance*:
 - An action a is relevant for a goal g if
 - $\Rightarrow a$ makes at least one of g 's literals true
 - $\rightsquigarrow g \cap effects(a) \neq \emptyset$
 - $\Rightarrow a$ does not make any of g 's literals false
 - $\rightsquigarrow g^+ \cap effects^-(a) = \emptyset$ and $g^- \cap effects^+(a) = \emptyset$

Inverse State Transitions

- If a is relevant for g , then

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

- Otherwise $\gamma^{-1}(g, a)$ is undefined

- Example: suppose that

$$g = \{\text{on}(b1, b2), \text{on}(b2, b3)\}$$

$$a = \text{stack}(b1, b2)$$

- What is $\gamma^{-1}(g, a)$?

Backward-Search

Algorithm 2 Backward-search(O, s_0, g)

$\pi \leftarrow$ the empty plan

loop

 if s_0 satisfies g then return π

$A \leftarrow \{a \mid a \text{ is a ground instance of an operator in } O \text{ and } \gamma^{-1}(g, a) \text{ is defined}\}$

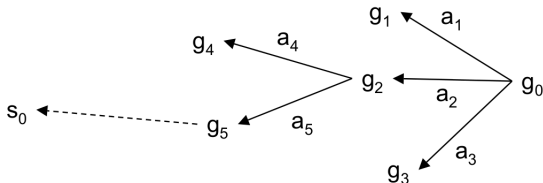
 if $A = \emptyset$ then return failure

 non-deterministically choose any action $a \in A$

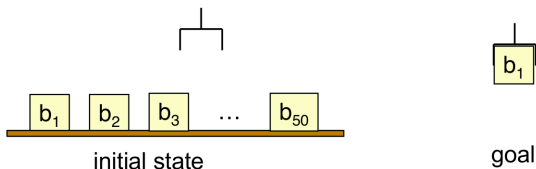
$\pi \leftarrow a.\pi$

$g \leftarrow \gamma^{-1}(g, a)$

end loop



Efficiency of Backward Search

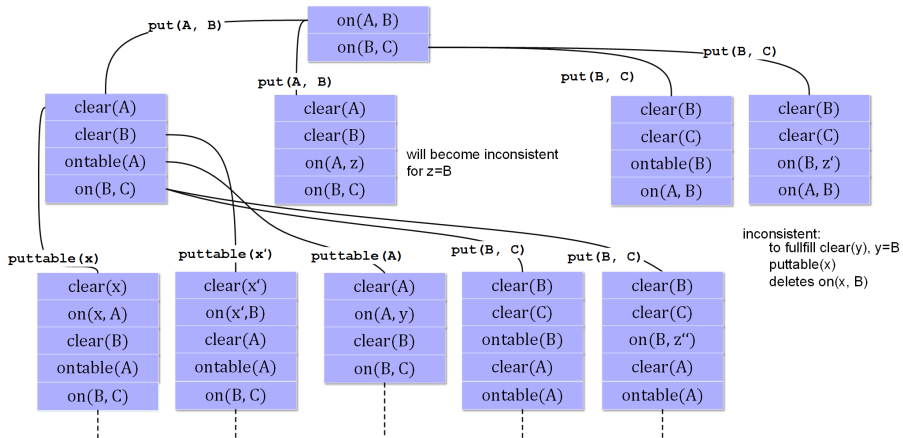


- Backward search can *also* have a very large branching factor
 - E.g., an operator o that is relevant for g may have many ground instances a_1, a_2, \dots, a_n such that each a_i 's input state might be unreachable from the initial state
- As before, deterministic implementations can waste lots of time trying all of them

Remarks on Backward Planning

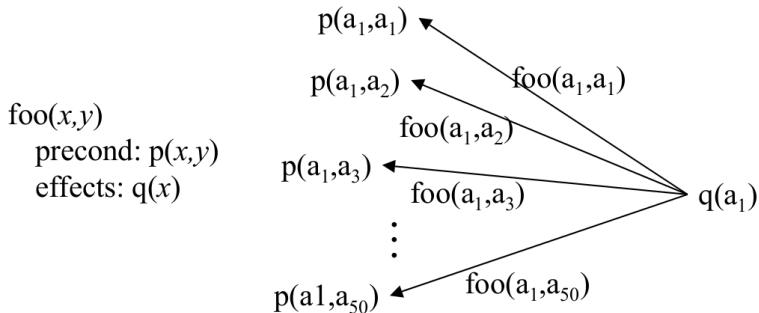
- Forward search also called *progression* planning
- Backwards search also called *regression* planning
- Problem with backwards planning:
inconsistent states can be produced (see blocksworld example)
- Compare Graphplan strategy:
build a Planning Graph by forwards search (polynomial effort) and
extract the plan from the graph backwards (exponential effort, as
usual for planning)

Backward Planning cont.

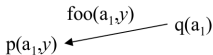


Axiom: $\forall x, y \text{ on}(x, y) \rightarrow \neg \text{clear}(y)$

Lifting



- Can reduce the branching factor of backward search if we partially instantiate the operators
 - this is called *lifting*



Lifted Backward Search

- More complicated than Backward-search
 - Have to keep track of what substitutions were performed
- But it has a much smaller branching factor
- mgu = most general unifier (see later), e.g. for $foo(x, y)$, substitution $\theta = \{x \leftarrow a_1\}$ results in equality between all effects of $foo(a_1, y)$ and goal $q(a_1)$

Algorithm 3 Lifted-backward-search(O, s_0, g)

$\pi \leftarrow$ the empty plan

loop

if s_0 satisfies g then return π

$A \leftarrow \{(o, \theta) \mid o \text{ is a standardization of an operator in } O,$
 $\theta \text{ is an mgu for an atom of } g \text{ and an atom of } effects^+(o),$
 $\text{and } \gamma^{-1}(\theta(g), \theta(o)) \text{ is defined } \}$

if $A = \emptyset$ then return failure

non-deterministically choose a pair $(o, \theta) \in A$

$\pi \leftarrow$ the concatenation of $\theta(o)$ and $\theta(\pi)$

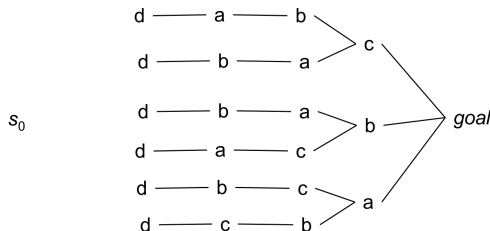
$g \leftarrow \gamma^{-1}\theta(g), \theta(o)$

end loop

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
 - Suppose actions a , b , and c are independent, action d must precede all of them, and there's no path from s_0 to d 's input state
 - We'll try all possible orderings of a , b , and c before realizing there is no solution
 - More about this in Chapter 5 (Plan-Space Planning)

in Ghallab, Malik, Dana Nau, and Paolo Traverso. Automated planning: theory & practice. Elsevier, 2004.

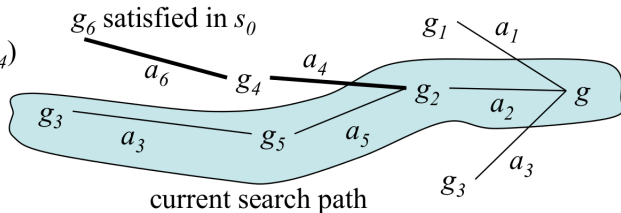


STRIPS

- $\pi \leftarrow$ the empty plan
- do a modified backward search from g
 - instead of $\gamma^{-1}(s, a)$, each new set of sub-goals is just $\text{precond}(a)$
 - 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

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

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



STRIPS

- by Fikes & Nilsson (1971),
“**S**tanford **R**esearch **I**nstitute **P**roblem **S**olver”
- classical example:
moving boxes between rooms (“Strips World”)
- Originally:
representation formalism (relying on CWA) and planning algorithm
today:
“STRIPS planning” refers to classical representation without
extensions and not to a specific algorithm
- STRIPS algorithm:
a **linear** (and therefore incomplete) approach
- compare to:
General Problem Solver (GPS), a cognitively motivated problem
solving algorithm which is also linear and therefore incomplete

STRIPS Algorithm

- Backward-search with a kind of hill climbing strategy
- In each recursive call only such sub-goals are relevant which are preconditions of the last operator added
- Consequence:
considerable reduction of branching, but resulting in incompleteness
- Linear planning:
organizing sub-goals in a stack
- Non-linear planning:
organizing sub-goals in a set, interleaving of goals

STRIPS Algorithm

Algorithm 4 STRIPS (O, s, g)

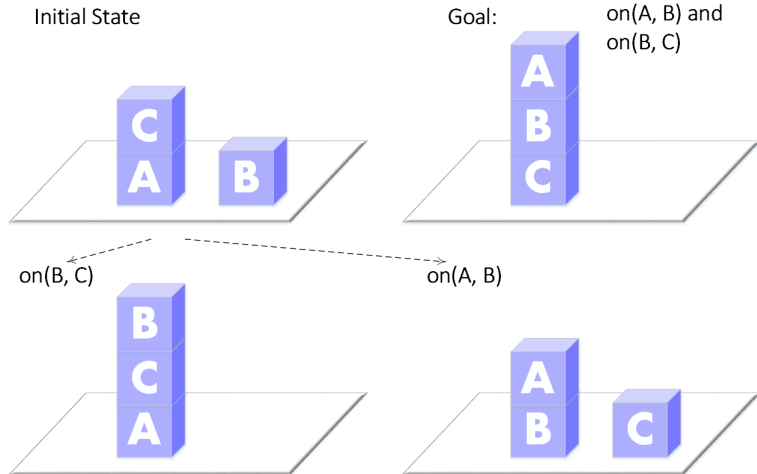
```

 $\pi \leftarrow$  empty plan
loop
  if  $s$  satisfies  $g$  then
    return  $\pi$ 
  end if
   $A \leftarrow \{a \mid a \text{ is a ground instance of an operator in } O, \text{ and } a \text{ is relevant for } g\}$ 
  if  $A = \emptyset$  then
    return failure
  end if
  non-deterministically choose any action  $a \in A$ 
   $\pi' \leftarrow$  STRIPS( $O, s, \text{precond}(a)$ )
  if  $\pi' = \text{failure}$  then
    return failure
    ;;if we get here, then  $\pi'$  achieves  $\text{precond}(a)$  from  $s$ 
  end if
   $s \leftarrow \gamma(s, \pi')$ 
  ;;s now satisfies  $\text{precond}(a)$ 
   $s \leftarrow \gamma(s, a)$ 
   $\pi \leftarrow \pi.\pi'.a$ 
end loop

```

Incompleteness of Linear Planning

The Sussman Anomaly



Sussman Anomaly

Linear planning corresponds to dealing with goals organized in a **stack**:

$[on(A, B), on(B, C)]$

try to satisfy goal $on(A, B)$

 solve sub-goals $[clear(A), clear(B)]$ ¹

 all sub-goals hold after $puttable(C)$

 apply $put(A, B)$

goal $on(A, B)$ is reached

try to satisfy goal $on(B, C)$.

¹We ignore the additional subgoal $ontable(A)$ resp. $on(A, z)$ here.

Interleaving of Goals

- Non-linear planning allows that a sequence of planning steps dealing with one goal is interrupted to deal with another goal.
- For the Sussman Anomaly, that means that after block C is put on the table pursuing goal $on(A, B)$, the planner switches to the goal $on(B, C)$.
- Non-linear planning corresponds to dealing with goals organized in a [set](#).
- The correct sequence of goals might not be found immediately without backtracking.

Interleaving of Goals cont.

$\{on(A, B), on(B, C)\}$

try to satisfy goal $on(A, B)$

$\{clear(A), clear(B), on(A, B), on(B, C)\}$

$clear(A)$ and $clear(B)$ hold after $puttable(C)$

try to satisfy goal $on(B, C)$

apply $put(B, C)$

try to satisfy goal $on(A, B)$

apply $put(A, B)$.

Rocket Domain

(Veloso)

- Objects:
 n boxes, Positions (Earth, Moon), one Rocket
- Operators:
load/unload a box, move the Rocket
(oneway: only from earth to moon, no way back!)
- Linear planning:
to reach the goal, that Box1 is on the Moon, load it, shoot the Rocket, unload it, now no other Box can be transported!

The Register Assignment Problem

- State-variable formulation:

Initial state: $\{\text{value}(r1)=3, \text{value}(r2)=5, \text{value}(r3)=0\}$

Goal: $\{\text{value}(r1)=5, \text{value}(r2)=3\}$

Operator: $\text{assign}(r, v, r', v')$
precond: $\text{value}(r)=v, \text{value}(r')=v'$
effects: $\text{value}(r)=v'$

- STRIPS cannot solve this problem at all

Use of Domain Specific Knowledge

- The Sussman Anomaly can also be handled by the usage of domain-specific knowledge

By Ghallab, Malik, Dana Nau, and Paolo Traverso. Automated planning: theory & practice. Elsevier, 2004.

- Example: block stacking using forward search

Quick Review of Blocks World

unstack(x,y)

Pre: $\text{on}(x,y)$, $\text{clear}(x)$, handempty

Eff: $\sim\text{on}(x,y)$, $\sim\text{clear}(x)$, $\sim\text{handempty}$,
 $\text{holding}(x)$, $\text{clear}(y)$

stack(x,y)

Pre: $\text{holding}(x)$, $\text{clear}(y)$

Eff: $\sim\text{holding}(x)$, $\sim\text{clear}(y)$,
 $\text{on}(x,y)$, $\text{clear}(x)$, handempty

pickup(x)

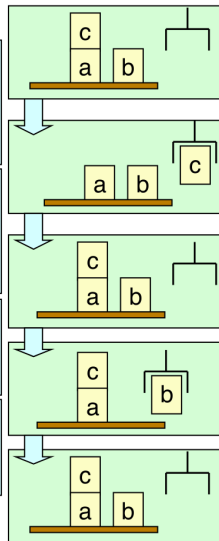
Pre: $\text{ontable}(x)$, $\text{clear}(x)$, handempty

Eff: $\sim\text{ontable}(x)$, $\sim\text{clear}(x)$, $\sim\text{handempty}$, $\text{holding}(x)$

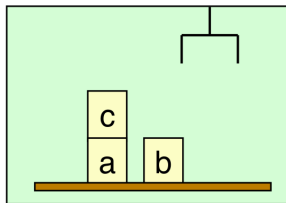
putdown(x)

Pre: $\text{holding}(x)$

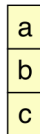
Eff: $\sim\text{holding}(x)$, $\text{ontable}(x)$, $\text{clear}(?x)$, handempty



The Sussman Anomaly



Initial state



goal

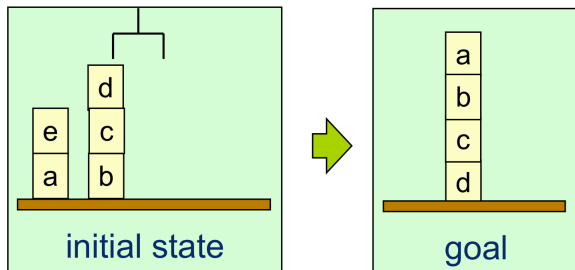
- On this problem, STRIPS can't produce an irredundant solution
 - Try it and see

Domain-Specific Knowledge

- A blocks-world planning problem $P = (O, s_0, g)$ is solvable if s_0 and g satisfy some simple consistency conditions
 - g should not mention any blocks not mentioned in s_0
 - 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) - see **a** below
 - s contains **on**(x,y) and g contains **ontable**(x) - see **d** below
 - s contains **on**(x,y) and g contains **on**(x,z) for some $y \neq z$
 \Rightarrow see **c** below
 - s contains **on**(x,y) and y needs to be moved - see **e** below



Domain-Specific Algorithm

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

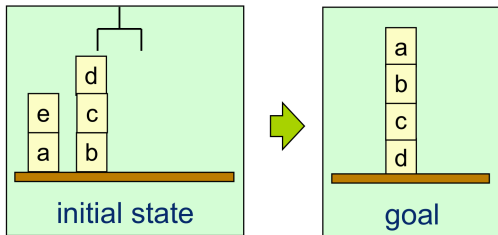
then move x to the table

else if the goal is satisfied

then return the plan

else return failure

repeat



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

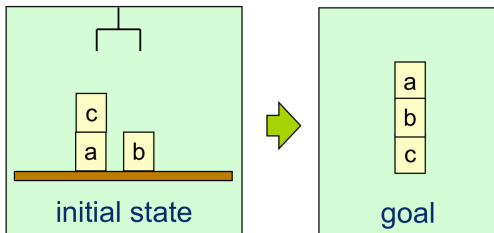
then move x to the table

else if the goal is satisfied

then return the plan

else return failure

repeat



Properties

- The block-stacking algorithm:
 - Sound, complete, guaranteed to terminate
 - Runs in time $O(n^3)$
 - ⇒ 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)
 - ⇒ *PLAN LENGTH* for the blocks world is NP- complete

Specific vs. General Approaches

- In general, it is more useful to have a general purpose approach, such as a domain-independent planner
- However, if there is knowledge available for a domain, it should not be ignored; but used to make the general approach more informed and thereby usually more efficient
- One possibility to exploit knowledge in a more general way, is to combine planning and machine learning

U. Schmid & E. Kitzelmann, Inductive Rule Learning on the Knowledge Level, *Cognitive Systems Research*, 12(3), 237-248, 2011.

Applying the inductive programming system IGOR2 to learn Tower building from solution examples.

Learning A Solution Strategy for BlocksWorld

Tower (9 examples of towers with up to four blocks, 1.2 sec)

(10 corresponding examples for Clear and IsTower as BK)

```
Tower(O, S) = S    if IsTower(O, S)
Tower(O, S) = put(O, Sub1(O, S),
                  Clear(O, Clear(Sub1(O, S),
                                  Tower(Sub1(O, S), S))))    if not(IsTower(O, S))
Sub1(s(O), S) = O .
```

Summary

- Planning is search
- Basic search techniques are forward (from initial state to a state fulfilling the goals) and backward (from the goals to the initial state)
- For backward search an inverse state-transition operator has to be defined
- Algorithms need to be sound and complete, furthermore, efficiency should be considered (branching factor during search!)
- A classical planning algorithm is Strips
- Strips is incomplete as demonstrated with the Sussman Anomaly
- Incompleteness can be overcome by defining domain specific algorithms