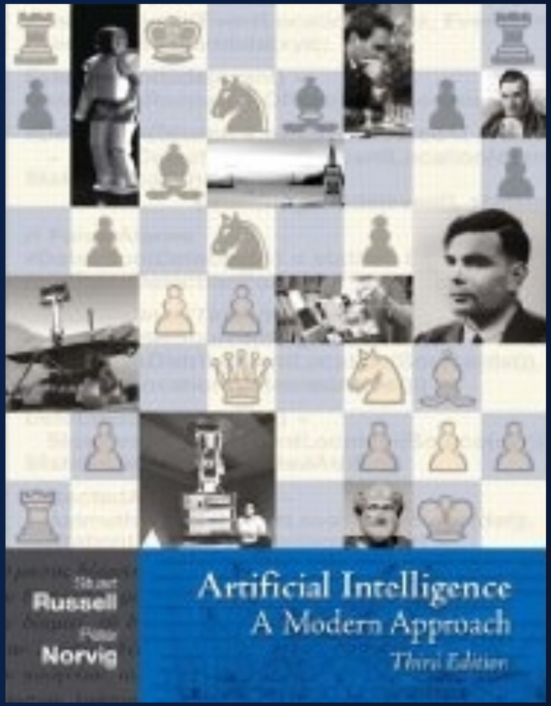


AI Fundamentals: planning

Maria Simi

IN SUPREMÆ DIGNITATIS
1343

Classical planning

LESSON 1: INTRODUCTION TO CLASSICAL PLANNING – PDDL – PROGRESSIVE AND REGRESSIVE PLANNING – HEURISTICS FOR PLANNING

Planning vs problem solving & theorem proving

Planning combines two major areas of AI: search and logic.

1. Problem solving (searching for a solution path in a state graph) is already a form of *planning*, but we assume atomic states and a simple goal function
2. We discussed how to model actions and change in FOL with the *situation calculus* and related problems such as the **frame** and **qualification** problems.

We will introduce a suitable language for describing planning problems.

1. Allowing richer descriptions of actions and states (including goals), special algorithms and heuristics can be developed.
2. Suitable limitations to circumvent the computational problems with a FOL representation.

Planning vs problem solving

A problem solving *agent*. Goals and actions are not explicitly represented.

- *Actions*: a function defining successors states
- *Goal*: test $Goal(s)$ -> $\{true, false\}$.
- *Planning*: to obtain, by heuristic search, a path in the state graph, leading from the initial state to a goal. Heuristics are domain dependent.

A planning agent

- Has an explicit representation of the goal, of actions and their effect
- It is able to inspect the goal and decompose it, and make abstractions
- It can work freely on the plan construction manipulating plans

A planning agent can be more efficient.

Planning as satisfiability in PROP

1. The SAT problem to solve is obtained by **generating** a [huge] propositional sentence (in clausal form) that includes:
   - $Init^0$, a collection of assertions about the initial state. Example: $L^0_{1,1}$
   - *Transition*^{1}, . . . , *Transition*^{t}, the *successor-state axioms* for all possible action at each time up to some maximum time t. Example: $HaveArrow^{t+1} \Leftrightarrow (HaveArrow^t \wedge \neg Shoot^t)$
   - State constraints like non ubiquity: $L^z_{x,y} \Rightarrow \neg L^z_{x',y'}$
   - Action exclusion axioms: $\neg A_i^t \vee \neg A_j^t$ for any pair of actions A_i and A_j and any time t
   - The assertion that the goal is achieved at time t: $HaveGold^t \wedge ClimbedOut^t$
2. Solve the problem with a SAT solver.
3. If a model is found at t, a plan is made from the actions with value $true$. Minimum plans can be found by trying with increasing values ot t.

Modern SAT-solving technology makes the approach feasible, for medium-size problems

SATPLAN

```
function SATPLAN(init, transition, goal, T_max) returns solution or failure
   inputs: init, transition, goal, constitute a description of the problem
           T_max, an upper limit for plan length

   for t = 0 to T_max do
      cnf ← TRANSLATE-TO-SAT(init, transition, goal, t)
      model ← SAT-SOLVER(cnf)
      if model is not null then
         return EXTRACT-SOLUTION(model)
   return failure
```

The planning problem is translated into a CNF sentence for increasing values of t until a solution is found or the upper limit to the plan length is reached.

The situation calculus

- A special ontology made out of situations, *fluents*, actions …
- Problems in defining actions and their effect (*frame problem*) partially solved by defining *state successor axioms*, one for each fluent. For example:

$$Clear(y, Result(a, s)) \Leftrightarrow$$
$$[On(x, y, s) \wedge Clear(x, s) \wedge Clear(z, s) \wedge x \neq z \wedge a = move(x, y, z))] \vee$$
$$[On(x, y, s) \wedge Clear(x, s) \wedge (a = unstack(x, y))] \vee$$
$$[Clear(y, s) \wedge (a \neq move(z, w, y)) \wedge (a \neq stack(z, y))]$$

- Effect of a sequence of actions: $Result$: $[A^*] \times S \rightarrow S$

1. $Result([\,], s) = s$
2. $Result([a \,|\, seq], s) = Result(seq, Result(a, s))$

Planning as theorem proving in FOL

Planning is the generatation of a sequence of actions p to reach the goal G. This amounts to proving that:

$$\exists p\ G(Result(p, s_0))$$

The Green planner used a theorem prover based on **resolution**.

The task is made complex by different sources of non determinism:

- the length of the sequence of actions is not known in advance
- *frame axioms* may infer many things that are irrelevant
- we need to resort to *ad hoc* strategies, completeness is not guaranteed

A general theorem prover is **inefficient** and **semi-decidable**

We do not have any guarantee on the efficiency of the generated plan.

Definition of classical planning

1. In classical planning we assume fully observable, deterministic, static environments with single agents.
2. We assume a **factored representation**: a state of the world is represented by a collection of variables.
3. **PDDL**, the **Planning Domain Definition Language** is a specialized language for describing planning problems and in particular:
   - States
   - Applicable actions ($ACTIONS(s)$) and transition model ($RESULT(s, a)$)
   - Goal states

PDDL: a language for planning

States: conjunction/set of **fluents,** ground positive atoms, no variables, no functions.

Examples: $At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$

Database semantics is used:

1. *Closed World Assumption*: fluents that are not mentioned are *false*
2. *Unique Name Assumption*: distinct names refer to distinct individuals

Actions: a set of action schemas that implicitly define the $Actions(s)$ and *Result*(*s*, *a*)

Actions are defined in a way that avoids the *frame problem* since they implicitly assume that you carefully specify all the changes and all the rest remains as it was before the action.

In most problems things that change are a few compared to the ones that do not.

PDDL: actions

Actions are defined by a set of **action schemas**, corresponding to parametric actions or operators.

Example of action schema, flying from a location to another one:

$Action(Fly(p, from, to),$ // p, $from$ and to are variables

$PRECOND: At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$

$EFFECT: \neg At(p, from) \land At(p, to))$

p, $from$, to are universally quantified variables, that need to be instantiated.

An instance:

$Action(Fly(P1, SFO, JFK),$

$PRECOND: At(John, SFO) \land Plane(P1) \land Airport(SFO) \land Airport(JFK)$

$EFFECT: \neg At(P1, SFO) \land At(P1, JFK))$

PDDL: preconditions and effects

$Action(Fly(p, from, to),$

$PRECOND: At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$

$EFFECT: \neg At(p, from) \land At(p, to))$

$PRECOND$ a list of **preconditions** to be *satisfied* in s for the action to be **applicable**

$(a \in ACTIONS(s)) \Leftrightarrow s \vDash PRECOND(a)$

i.e. positive literals are in s and negated literals are not in s

$EFFECT$ the successor state s' is obtained from s as result of the action by:

- adding positive effects to s — $add\ list,\ ADD(a)$
- removing negative effects from s — $delete\ list,\ DEL(a)$

$RESULT(s, a) = (s - DEL(a)) \cup ADD(a)$

Example: The action $Fly(P1, SFO, JFK)$ would remove $At(P1, SFO)$ and add $At(P1, JFK)$

PDDL: initial state and goal description

Initial state: a collection of ground positive atoms

Goal: a conjunction of literals (positive or negative) that may contain variables, for example $At(p, SFO) \wedge Plane(p)$ is the goal of having **any** plane to SFO.

The problem is solved when we can find a sequence of actions that end in a state s **that entails the goal**.

Example: air cargo transport

$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
$\wedge\ Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\wedge\ Airport(JFK) \wedge Airport(SFO))$
$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg At(c, a) \wedge In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c, a) \wedge \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p, from) \wedge At(p, to))$

Plan: $[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$
$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)]$

Example: spare tire

$Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))$
$Goal(At(Spare, Axle))$

$Action(Remove(obj, loc),$
  PRECOND: $At(obj, loc)$
  EFFECT: $\neg At(obj, loc) \land At(obj, Ground))$

$Action(PutOn(t, Axle),$
  PRECOND: $Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle)$
  EFFECT: $\neg At(t, Ground) \land At(t, Axle))$

$Action(LeaveOvernight,$
  PRECOND:
  EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
    $\land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))$

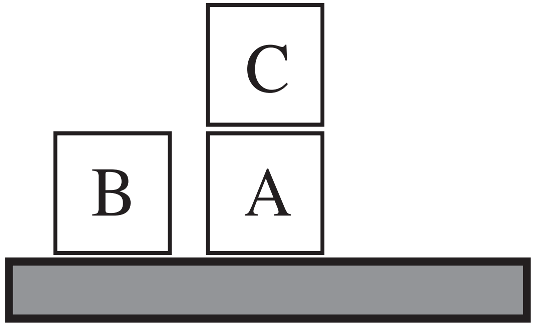
Plan: $[Remove(Flat, Axle), Remove(Spare, Trunk), PutOn(Spare, Axle)]$

Example: the blocks world

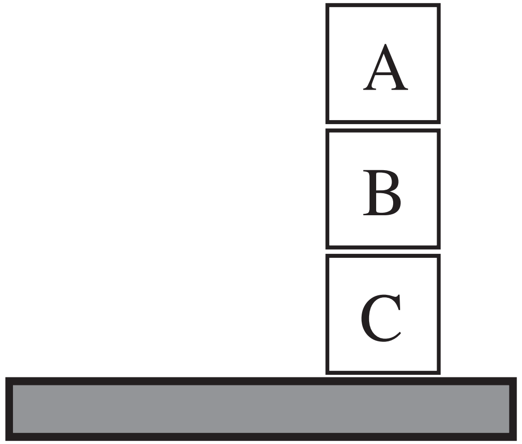
$$Init(On(A, Table) \land On(B, Table) \land On(C, A)$$
$$\land\ Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C))$$
$$Goal(On(A, B) \land On(B, C))$$

$Action(Move(b, x, y),$
  PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
    $(b \neq x) \land (b \neq y) \land (x \neq y),$
  EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$

$Action(MoveToTable(b, x),$
  PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land (b \neq x),$
  EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$



Start State



Goal State

Plan: [$MoveToTable\ (C, A), Move(B, Table, C), Move(A, Table, B)$]

Algorithms and heuristics for planning

Decision problems for planning and complexity

1. PlanSAT: does a plan exist?
2. Bounded PlanSAT: there is a solution of length k or less?

Decidability: both problems are decidable for classical planning.

- PlanSAT **without functions,** since the number of states is finite.
- Bounded PlanSAT: always decidable, also with functions.

Complexity results:

- if we disallow negative effects, both problems are still NP-hard.
- if we also disallow negative preconditions, PlanSAT reduces to the class P.
- For many domains (including the blocks world and the air cargo world), Bounded PlanSAT is NP-complete while PlanSAT is in P.

Planning as state-space search

In **planning graphs** nodes are states arcs are actions.

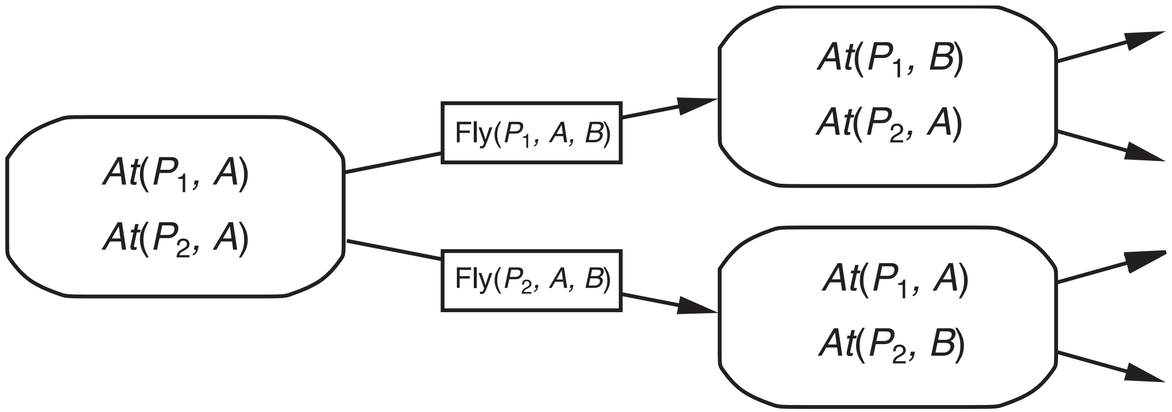
1. **Progression planning**: *forward* search from the initial state to the goal state.

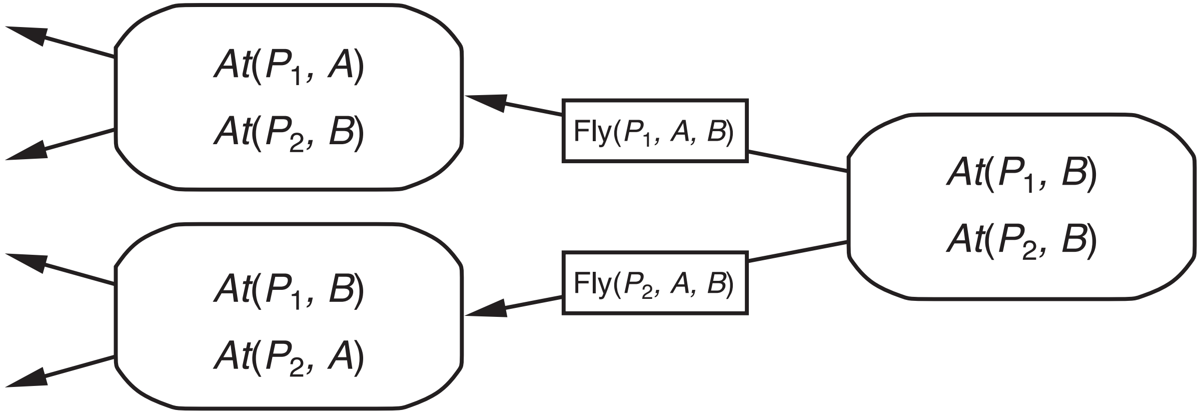
   Believed to be inefficient

   - prone to exploring *irrelevant* actions
   - planning problems often have large state spaces

2. **Regression planning**: backward search backward from the goal state to the initial state.

   The first approach attempted (STRIPS)





Regression planning

We start with the goal, a conjunction of literals, describing a set of worlds.

The PDDL representation allows to *regress* actions, i.e. to find a state g' from where it is possible to rich the goal with some action *a*:

$g' = (g - ADD(a)) \cup Precond(a)$ we do not say anything about $DEL(a)$

Regressing an action with variables, having goal $At\,(\mathrm{C2}, \mathrm{SFO})$:

$Action(Unload(\mathrm{C2}, p', \mathrm{SFO}),$ p' after renaming

$PRECOND: In(\mathrm{C2}, p') \wedge At(p, \mathrm{SFO}) \wedge Cargo(\mathrm{C2}) \wedge Plane(p') \wedge Airport(\mathrm{SFO})$

$\mathrm{EFFECT}: At\,(\mathrm{C2}, \mathrm{SFO}) \wedge \neg In\,(\mathrm{C2}, p')$

The regressed state description after $Unload(\mathrm{C2}, p', \mathrm{SFO})$, is

$g' = In(\mathrm{C2}, p') \wedge At(p', \mathrm{SFO}) \wedge Cargo(\mathrm{C2}) \wedge Plane(p') \wedge Airport(\mathrm{SFO})$

Relevant actions

Actions that can be used to reach a goal are defined **relevant**.

Relevant actions contribute to the goal but **must not have effects that negate some element of the goal**.

Several $Unload$ actions may be relevant but we choose the less instantiated one $Unload(\mathrm{C2}, p', \mathrm{SFO})$. Any plane will do. This is obtained by taking the **Most General Unifier** (θ).

Formally:

Assume a goal g containing a literal g_i and an action schema A, standardized to A'.

If A has an effect literal e_j where $Unify(g_i, e_j)=\theta$ and $a = SUBST(\theta, A)$ and if there is no effect in a that is the negation of a literal in g, then a is a **relevant** action towards g.

One of the earlier planning system was a **linear** regression planner called STRIPS (Fikes and Nilsson, 1971).

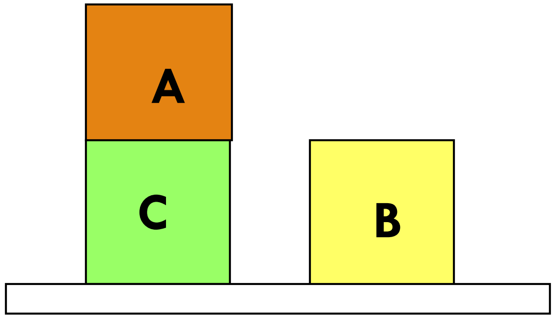
STRIPS by example

$Action(Stack(x, y),$
  $PRECOND: Clear(x) \land Table(x) \land Clear(y)$
  $EFFECT: On(\mathrm{x, y}) \land \neg Table(x) \land \neg Clear(y)$
)

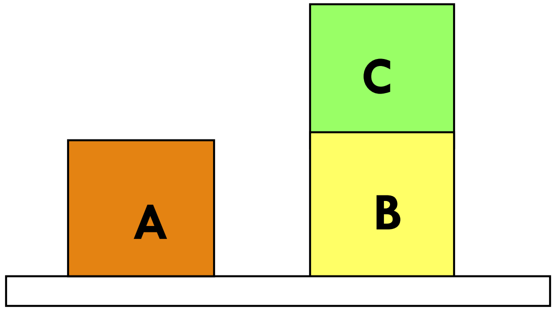
$Action(Unstack(x, y),$
  $PRECOND: Clear(x) \land On(x, y)$
  $EFFECT: \neg On(\mathrm{x, y}) \land Table(x) \land Clear(y)$
)

Initial state: $On(\mathrm{A, C}) \land Table(B) \land Table(C)$

Goal: $On(\mathrm{C, B}) \land Table(A) \land Table(B)$

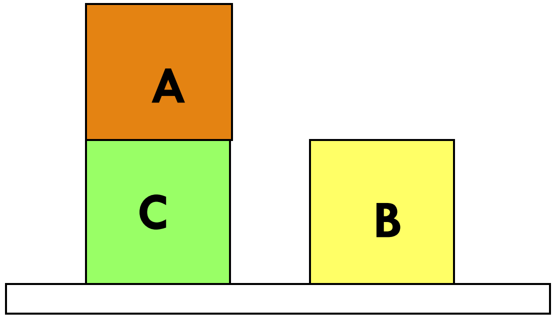


Initial state

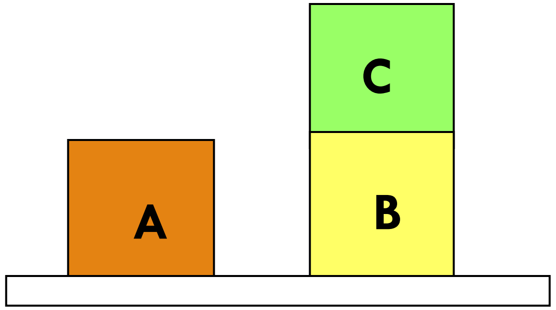


Goal state

STRIPS in action



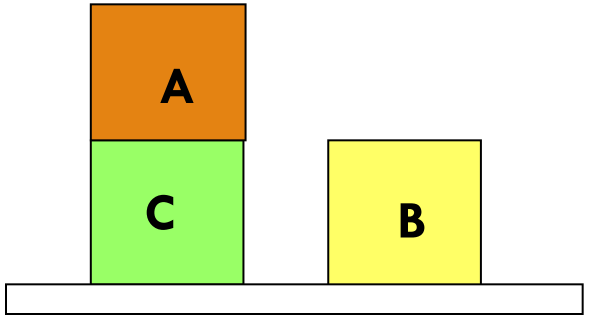
Initial state



Goal state

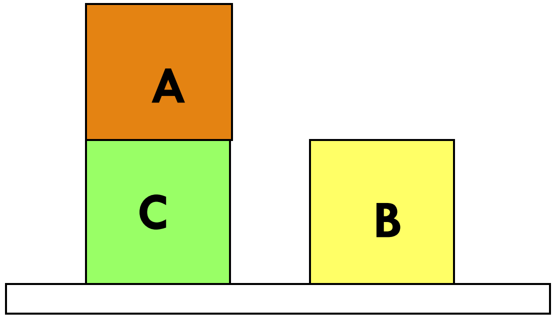
STACK

$On(\mathrm{C}, \mathrm{B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$

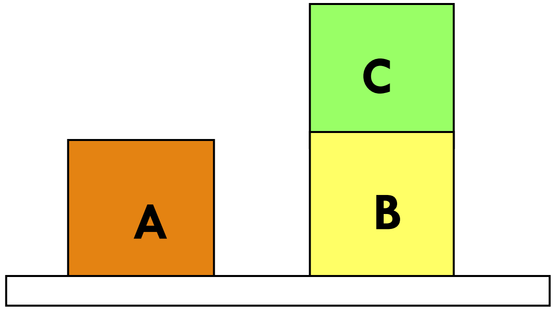


Current state

STRIPS in action



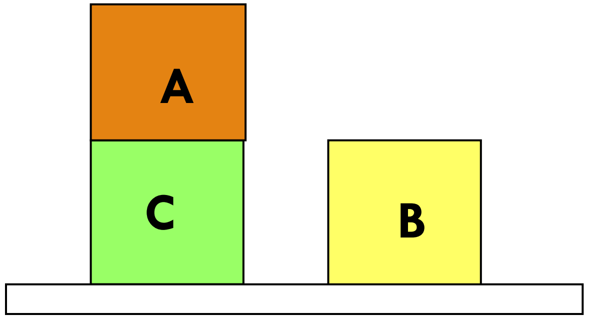
Initial state



Goal state

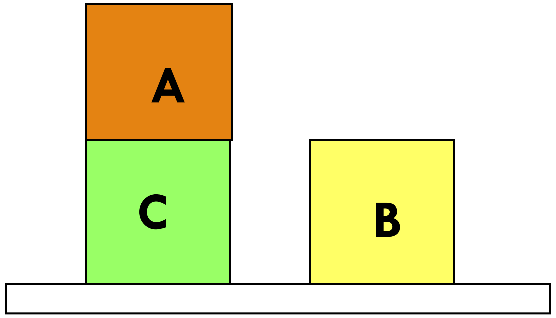
STACK

$$Table\,(\mathrm{B})$$
$$Table(\mathrm{A})$$
$$On(\mathrm{C},\mathrm{B})$$
$$On(\mathrm{C},\mathrm{B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$$

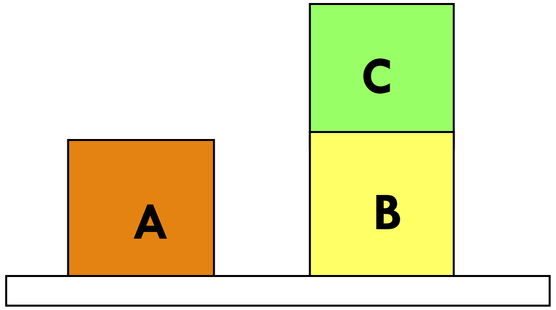


Current state

STRIPS in action



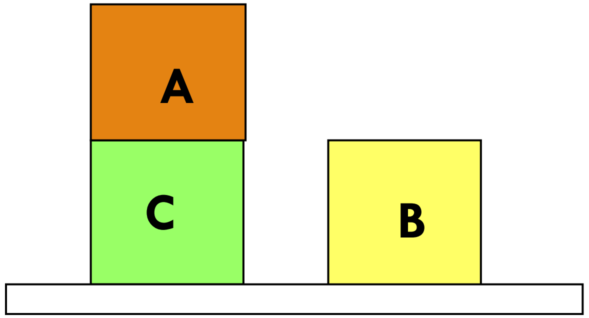
Initial state



Goal state

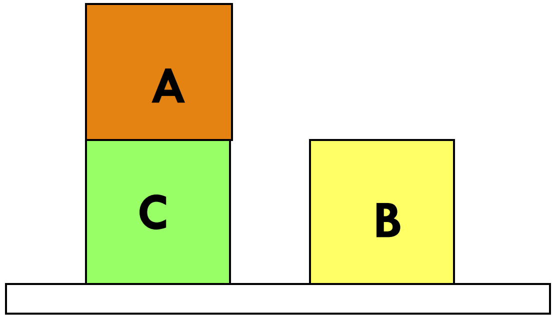
STACK

$$Table(\mathrm{A})$$
$$On(\mathrm{C, B})$$
$$On(\mathrm{C, B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$$

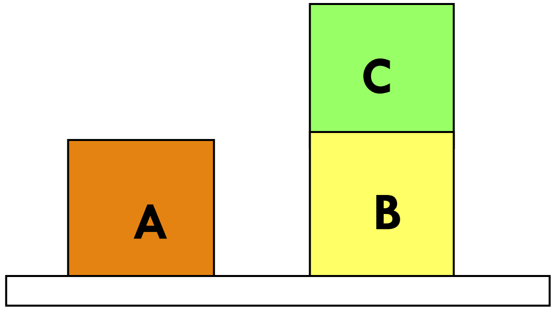


Current state

STRIPS in action



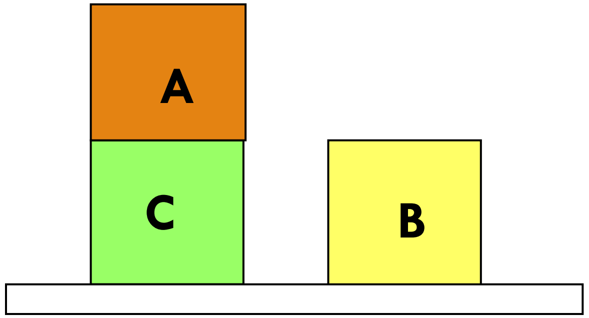
Initial state



Goal state

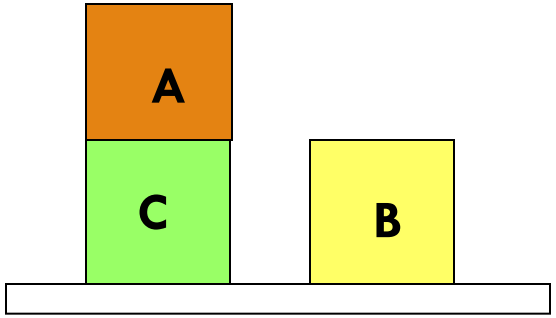
STACK

$$Clear(\mathrm{A})$$
$$Unstack(\mathrm{A})$$
$$Table(\mathrm{A})$$
$$On(\mathrm{C, B})$$
$$On(\mathrm{C, B}) \wedge Table(\mathrm{A}) \wedge Table\ (\mathrm{B})$$

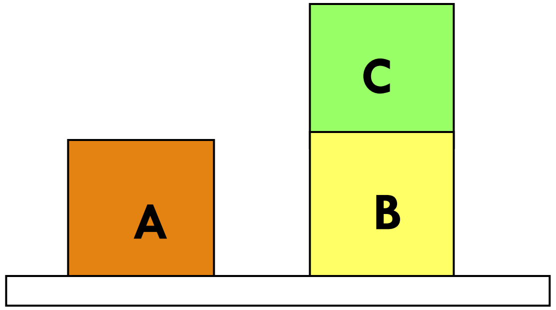


Current state

STRIPS in action



Initial state



Goal state

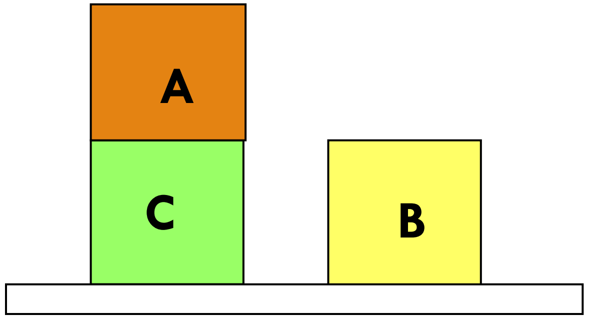
STACK

$\color{red}{Unstack}(\mathrm{A})$

$Table(\mathrm{A})$

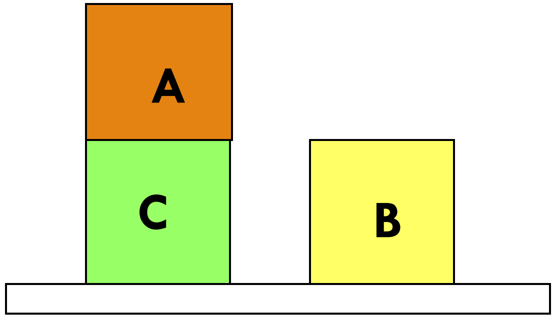
$On(\mathrm{C}, \mathrm{B})$

$On(\mathrm{C}, \mathrm{B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$

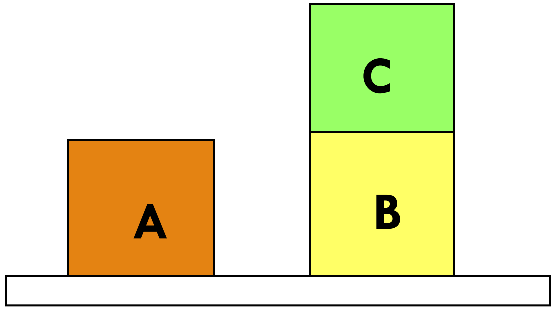


Current state

STRIPS in action



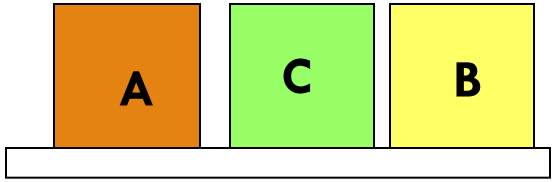
Initial state



Goal state

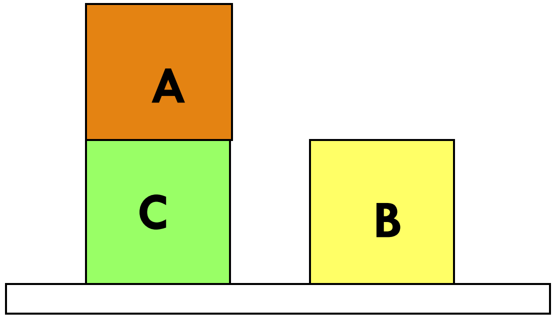
STACK

$$Table(\mathrm{A})$$
$$On(\mathrm{C, B})$$
$$On(\mathrm{C, B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$$

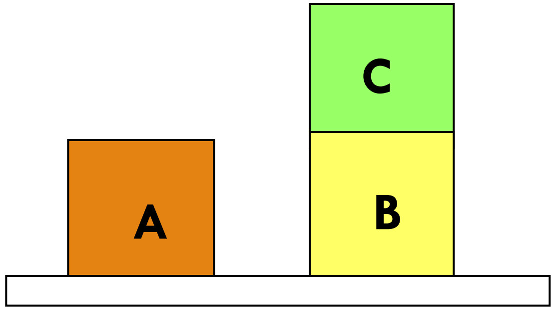


Current state

STRIPS in action



Initial state

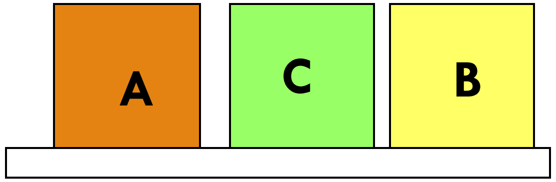


Goal state

STACK

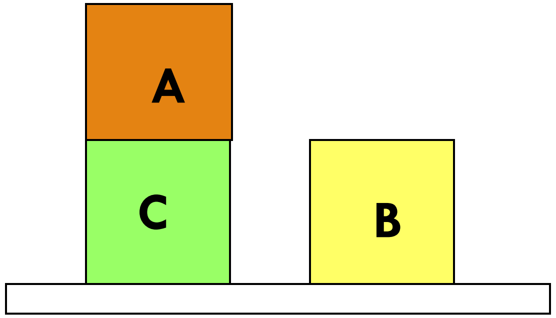
$$On(\mathrm{C}, \mathrm{B})$$

$$On(\mathrm{C}, \mathrm{B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$$

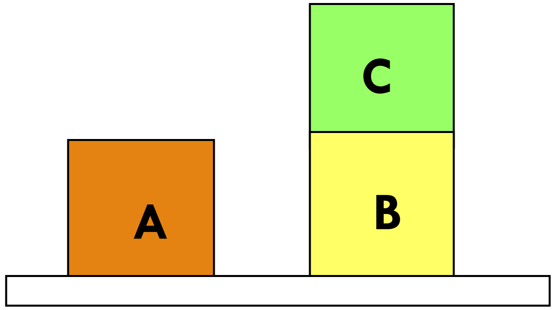


Current state

STRIPS in action



Initial state



Goal state

STACK

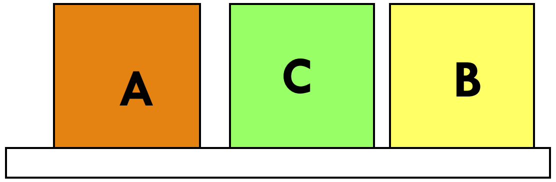
$Clear(\mathrm{B})$

$Clear(\mathrm{C})$

$Stack(\mathrm{C, B})$

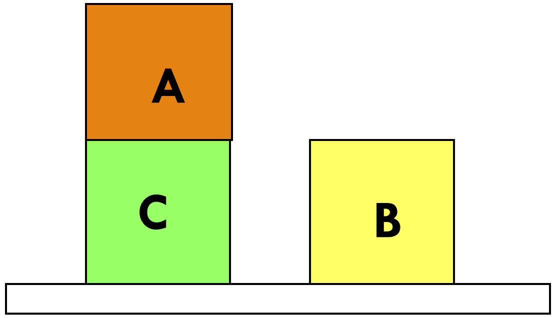
$On(\mathrm{C, B})$

$On(\mathrm{C, B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$

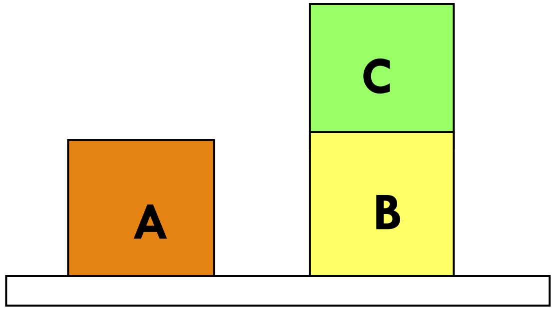


Current state

STRIPS in action



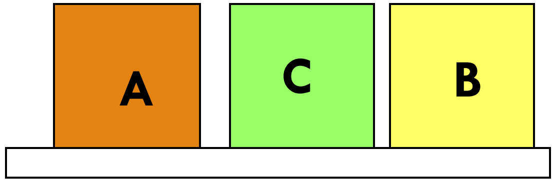
Initial state



Goal state

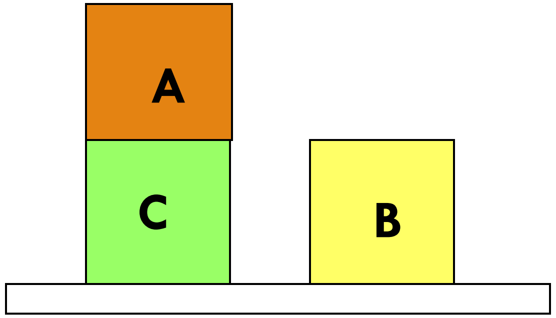
STACK

$$\mathit{Clear}(\mathrm{C})$$
$$\color{red}{\mathit{Stack}(\mathrm{C}, \mathrm{B})}$$
$$\mathit{On}(\mathrm{C}, \mathrm{B})$$
$$\mathit{On}(\mathrm{C}, \mathrm{B}) \wedge \mathit{Table}(\mathrm{A}) \wedge \mathit{Table}(\mathrm{B})$$

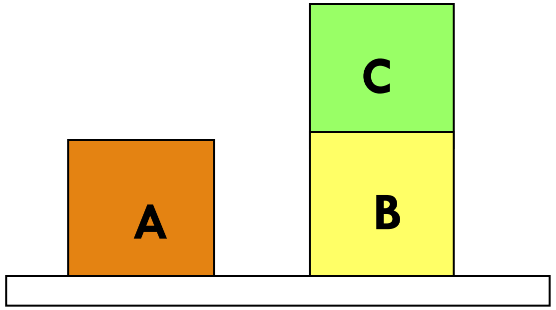


Current state

STRIPS in action



Initial state



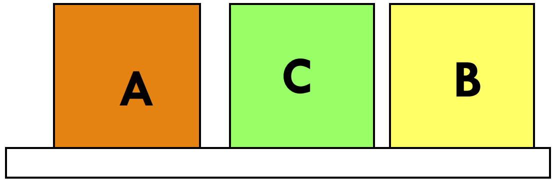
Goal state

STACK

$\textcolor{red}{Stack}(\mathrm{C}, \mathrm{B})$

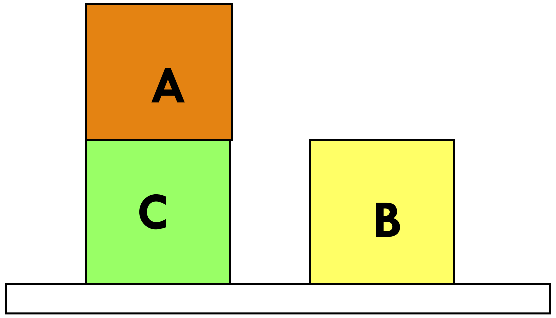
$On(\mathrm{C}, \mathrm{B})$

$On(\mathrm{C}, \mathrm{B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$

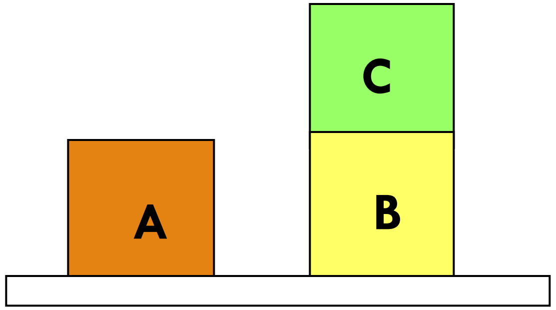


Current state

STRIPS in action



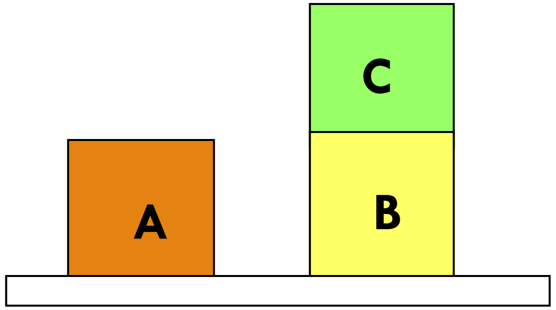
Initial state



Goal state

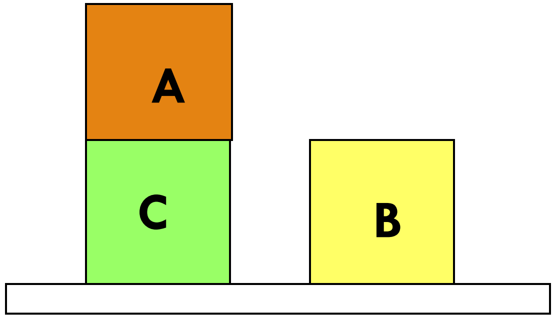
STACK

$$On(\mathrm{C, B})$$
$$On(\mathrm{C, B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$$

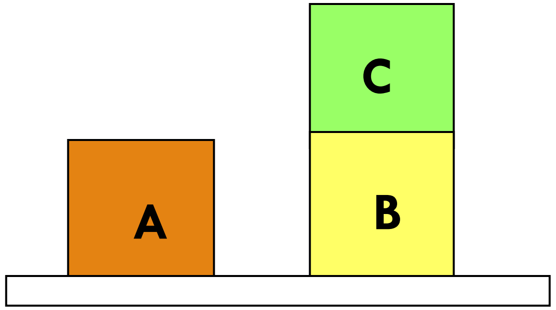


Current state

STRIPS in action

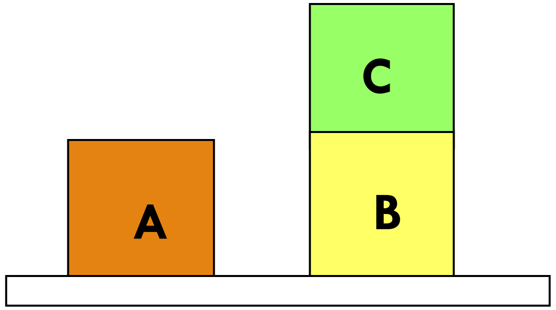


Initial state



Goal state

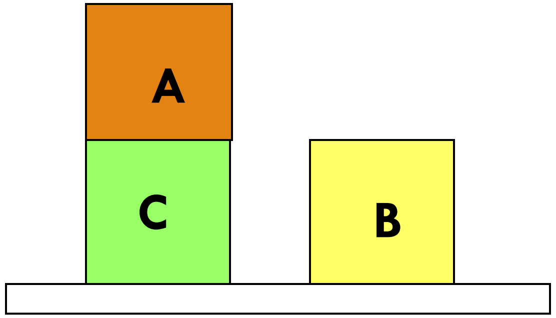
STACK



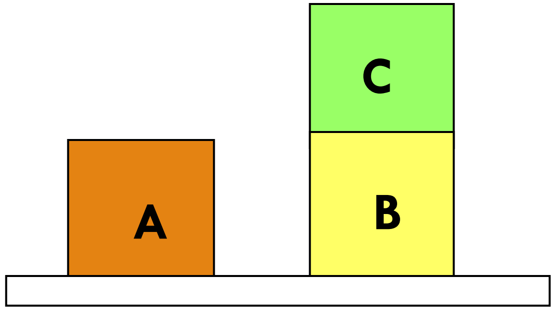
Current state

$$On(\mathrm{C}, \mathrm{B}) \wedge Table(\mathrm{A}) \wedge Table\,(\mathrm{B})$$

STRIPS in action

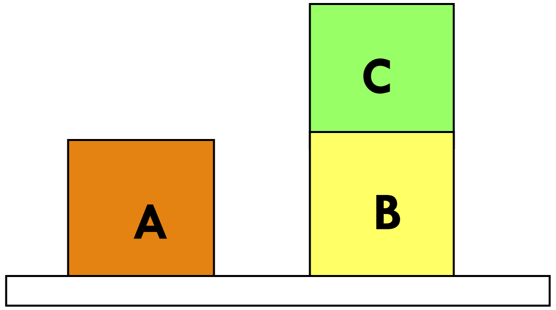


Initial state



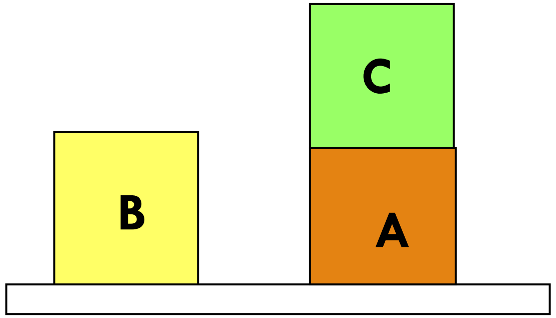
Goal state

STACK

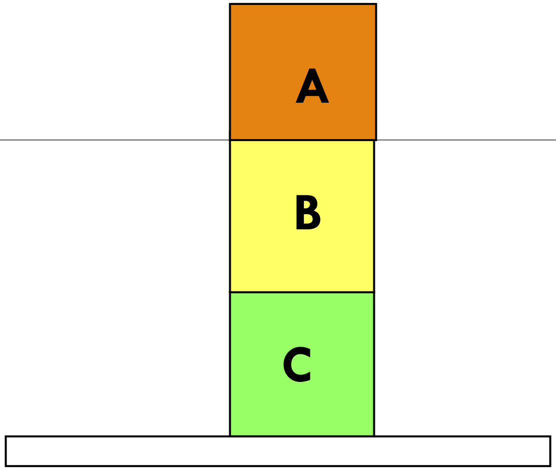


Current state

The Sussman anomaly



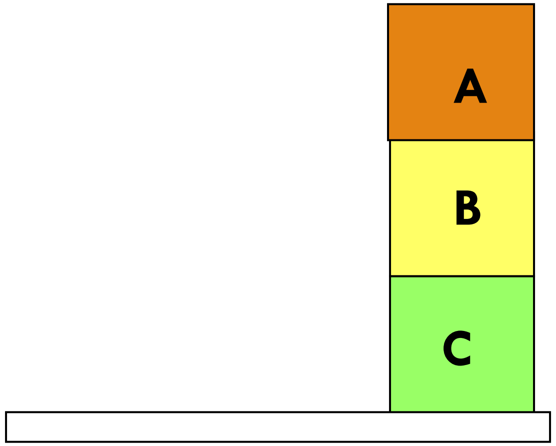
Initial state



Goal state

STACK

$On(b, c) \wedge On(a, b)$



The Sussman anomaly

The plan generated is:

$$[\mathit{Unstack}(\mathrm{C}), \mathit{Stack}(\mathrm{A}, \mathrm{B}), \mathit{Unstack}(\mathrm{A}), \mathit{Stack}(\mathrm{B}, \mathrm{C}), \mathit{Stack}(\mathrm{A}, \mathrm{B})]$$

Another plan that could have been generated stacking subgoals in a different order is:

$$[\mathit{Stack}(\mathrm{B}, \mathrm{C}), \mathit{Unstack}(\mathrm{B}), \mathit{Unstack}(\mathrm{C}),$$
$$\mathit{Stack}(\mathrm{A}, \mathbf{B}), \mathit{Unstack}(\mathrm{A}), \mathit{Stack}(\mathrm{B}, \mathrm{C}), \mathit{Stack}(\mathrm{A}, \mathbf{B})]$$

The ideal plan, that cannot obtained with **linear planning** (the goals interfere and cannot be achieved without interleaving actions) is:

$$[\mathit{Unstack}(\mathrm{C}), \mathit{Stack}(\mathrm{B}, \mathrm{C}), \mathit{Stack}(\mathrm{A}, \mathrm{B})]$$

Heuristics for planning

Problem relaxation is a common technique for finding admissible heuristics.

Given the factored representation we can devise general heuristics for planning.

Relaxing the problem. It can be done in two ways:

1. Adding arcs to the planning graph, thus making the graph easier to search:
   - Ignore preconditions heuristics
   - ignore delete lists heuristic
2. Clustering nodes, i.e. **state abstraction**.
   - Ignore some fluents

Decompose the problem by assuming **subgoal independence.**

Using a data structure called **'planning graphs'** (next lesson).

‘Ignore preconditions’ heuristics

The **ignore preconditions heuristic** drops all preconditions from actions, so every action is applicable in any state, any single goal literal can be satisfied in one step or no solution.

The number of steps to solve the goal approximated by the number of unsatisfied subgoals, but …

a. one action may satisfy more than one subgoal (non admissible estimate)
b. one action may undo the effect of another one (admissible)

An accurate heuristics is the following:

1. remove all preconditions and all effects except those that are literals in the goal
2. count the minimum number of actions required such that the union of those actions’ effects satisfies the goal (a problem of **set-cover**). NP-hard but greedy approximations exist.

As an alternative we could ignore only **some preconditions** from the actions.

Example: in the sliding tiles puzzle, removing the precondition of *empty destination*, leads to the Manhattan distance heuristics.

‘Ignore delete list’ heuristic

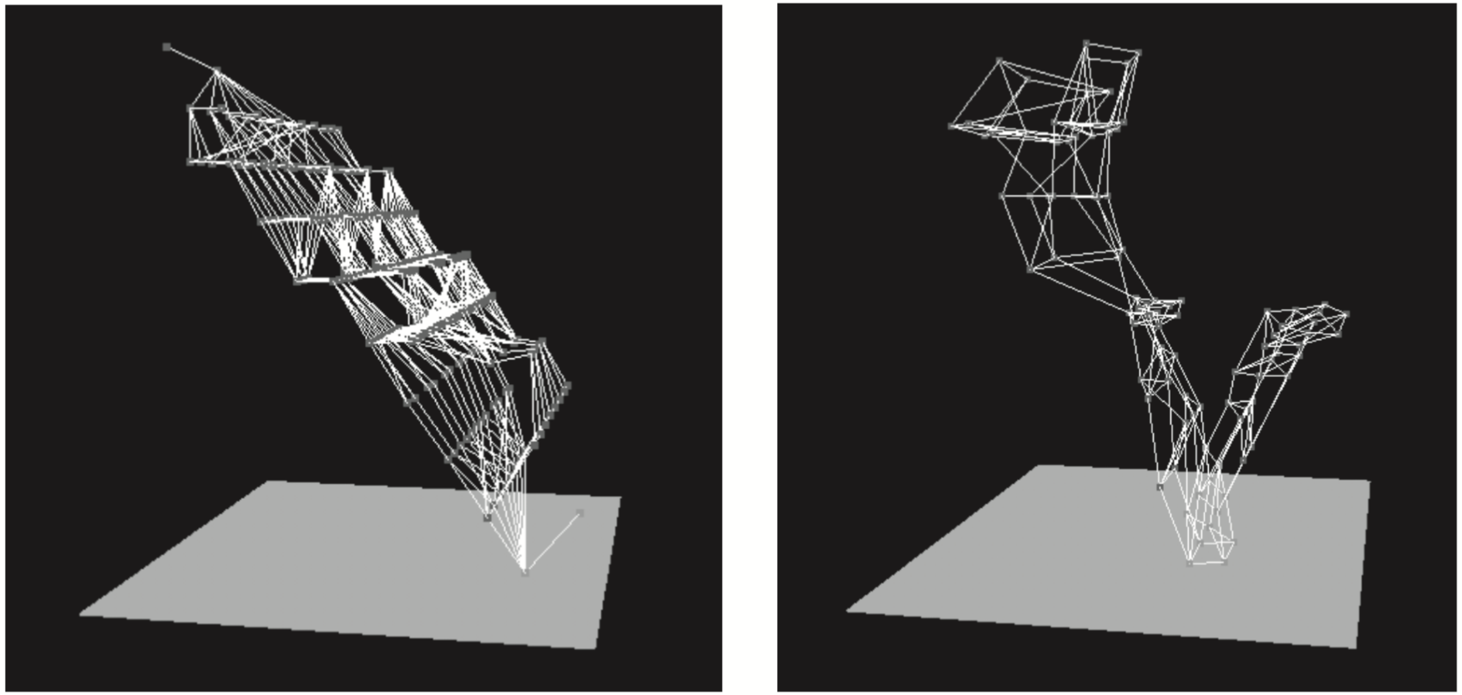
Assume that all goals and preconditions contain **only positive literals.**

Remove the delete lists from all actions (i.e., removing all negative literals from effects).

No action will ever undo the effect of actions, so there is a monotonic progress towards the goal.

Still NP-hard to find the exact solution of the relaxed problem but this can be approximated in P time, with *hill-climbing*. This strategy can be effective for some problems.

The ‘ignore delete-list’ heuristic in action



Two state spaces from planning problems with the **ignore-delete-lists heuristic**.
The height above the bottom plane is the heuristic score of a state; states on the bottom plane are goals. There are no local minima, so search for the goal is straightforward.
From Hoffmann (2005).

State abstraction

In order to reduce the number of states, we need other forms of relaxations, i.e. **state abstractions**.

A state abstraction is a many-to-one mapping from states in the ground/original representation of the problem to the abstract representation.

Common strategy: **ignore some fluents.**

Air cargo example: 10 airports, 50 planes, and 200 pieces of cargo.

There are $50^{10} \times 200^{50+10} \approx 10^{155}$ states to consider.

Consider a specific problem in that domain where all the packages are at 5 of the airports, and all packages at a given airport have the same destination.

Drop all the At fluents except for the ones involving one plane and one package at each of the 5 airports.

The cost of the solution to this smaller problem is an admissible heuristic.

Decomposition

Decomposition: divide a problem into parts, solve each part independently, and then combine the parts.

The subgoal **independence assumption** is that the cost of solving a conjunction of subgoals is approximated by the **sum** of the costs of solving each subgoal *independently*. This assumption can be:

1. *optimistic* (admissible), when there are negative interactions
2. *pessimistic*, and therefore inadmissible, when subplans contain redundant actions

Goal G, divided into disjoint subsets of fluents $G_1, \dots, G_n$

$P_1, \dots, P_n$ are plans solving the corresponding subgoals

$Cost(P_i)$ is a heuristic estimate of plan P_i

$max_i \, Cost(P_i)$ is an admissible heuristic while $\sum_i Cost(P_i)$ is not.

Conclusions

- Classical planning systems assume a factored representation of states and goals (PDDL) that makes possible specialized strategies and heuristics.
- We also discussed other approaches such as planning as SAT problem, and the limits of **linear** regression planning à la STRIPS.
- We discussed several strategies for finding admissible heuristics.
- Next time we will discuss **planning graphs**, a data structure that can be used to find better heuristics for forward planners, and is the basis for the **GraphPlan** algorithm.

Your turn

- ✓ Solve a new planning problem with SATplan: for example "*Monkey and bananas*", or the "*Shakey world*" (see descriptions at the end of AIMA Chapter 10).
- ✓ Solve a new planning problem with a regression planner such as STRIPS.
- ✓ An example of a system that makes use of effective heuristics is FF, or FastForward (Hoffmann, 2005). Discuss.
- ✓ Discuss heuristics that you find in the literature.

References

- Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach* (3rd edition). Pearson Education 2010 [Chapter 4, 10]