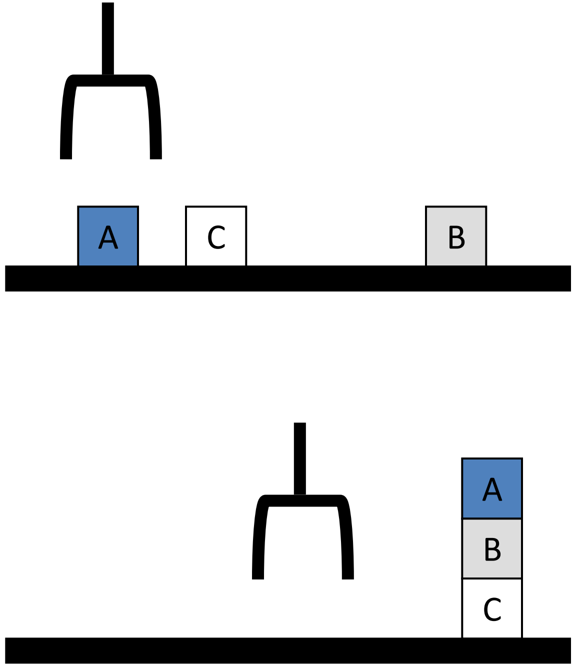
Planning

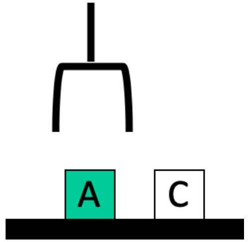
CMSC 471

Blocks World Planning



Blocks world

The blocks world is a micro-world with a table, a set of blocks, and a robot hand



Some constraints for a simple model:

- Only one block can be on another block
- Any number of blocks can be on the table
- The hand can only hold one block

Typical representation uses a logic notation:

ontable(b) ontable(d)

on(c,d) holding(a)

clear(b) clear(c)

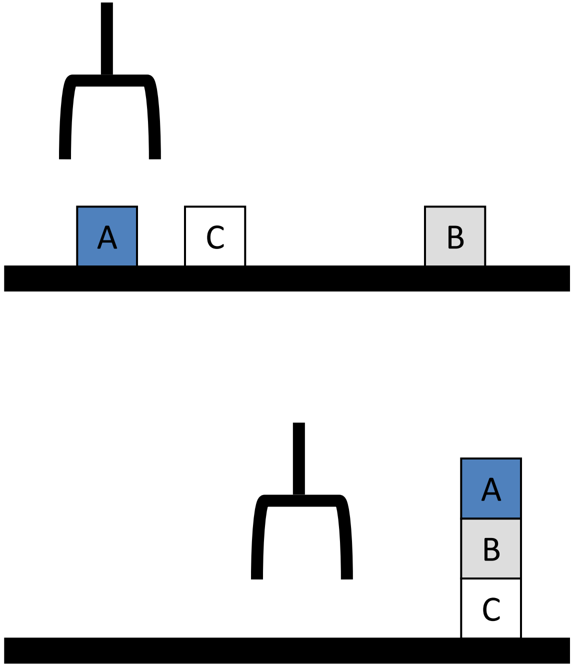
Typical BW planning problem

Initial state:

- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:

- on(b,c)
- on(a,b)
- ontable(c)



Typical BW planning problem

Logical assertions describing initial & final states

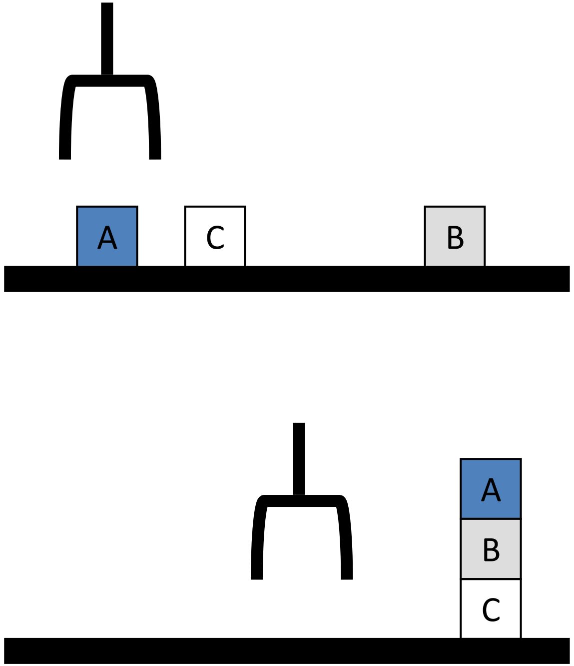
Sequence of robot actions

Initial state:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Goal state:

on(b,c)
on(a,b)
ontable(c)



Plan:

pickup(b)
stack(b,c)
pickup(a)
stack(a,b)

Planning problem

- Find sequence of actions to achieve goal state when executed from initial state given
  - set of possible primitive actions, including their *preconditions* and *effects*
  - initial state description
  - goal state description
- Compute plan as a sequence of actions that, when executed in initial state, achieves goal state
- States specified as a KB , i.e. conjunction of conditions
  - e.g., $ontable(a) \wedge on(b, a)$

Planning vs. problem solving

- Problem solving methods can solve similar problems
- Planning is more powerful and efficient because of the representations and methods used
- States, goals, and actions are decomposed into sets of sentences (usually in first-order logic)
- Search often proceeds through *plan space* rather than *state space* (though there are also state-space planners)
- Sub-goals can be planned independently, reducing the complexity of the planning problem

Typical simplifying assumptions

- Atomic time: Each action is indivisible
- No concurrent actions: but actions need not be ordered w.r.t. each other in the plan
- Deterministic actions: action results completely determined — no uncertainty in their effects
- Agent is the sole cause of change in the world
- Agent is omniscient with complete knowledge of the state of the world
- Closed world assumption: everything known to be true in world is included in state description and anything not listed is false

Blocks world

The blocks world consists of a table, a set of blocks and a robot hand

Some domain constraints:

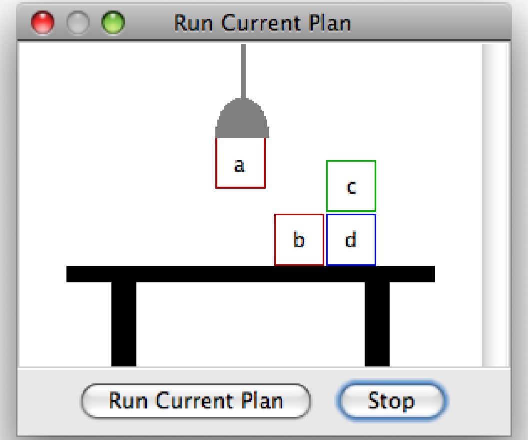
- Only one block can be on another block
- Any number of blocks can be on the table
- The hand can only hold one block

Typical representation:

ontable(b) ontable(d)

on(c,d) holding(a)

clear(b) clear(c)



Meant to be a simple model!

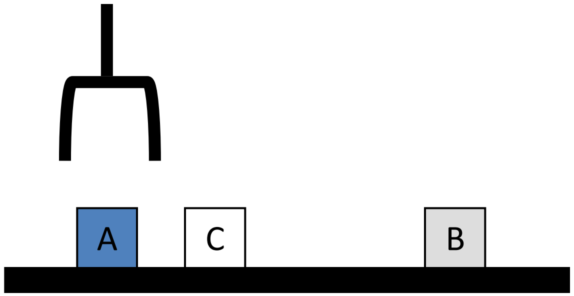
Typical BW planning problem

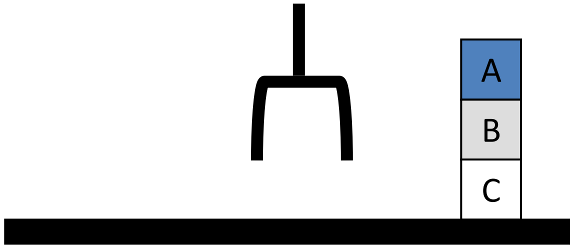
Initial state:

- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:

- on(b,c)
- on(a,b)
- ontable(c)





A plan:

- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

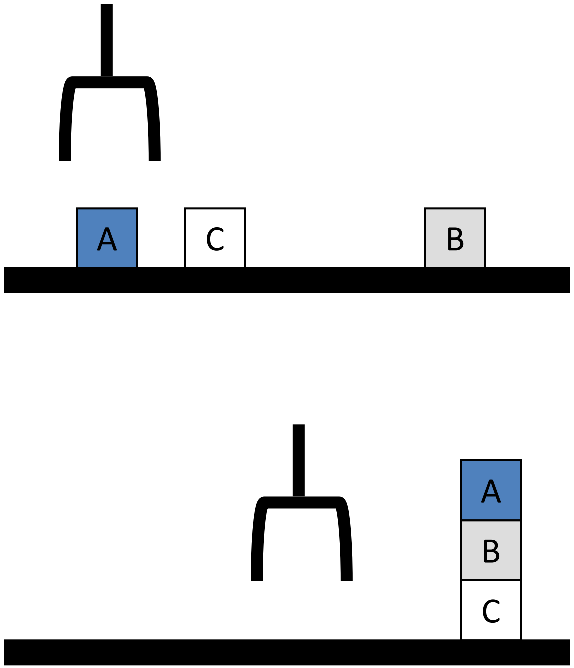
Typical BW planning problem

Initial state:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Goal:

on(a,b)
on(b,c)
ontable(c)



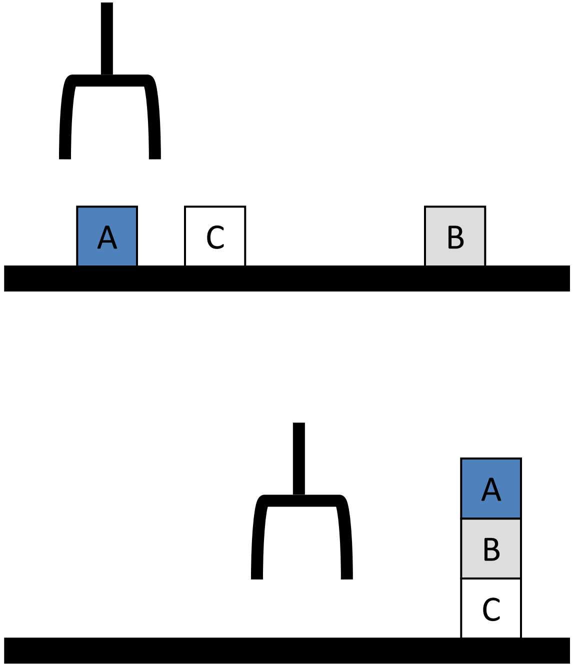
A plan:

pickup(a)
stack(a,b)
unstack(a,b)
putdown(a)
pickup(b)
stack(b,c)
pickup(a)
stack(a,b)

Typical BW planning problem

Initial state:
- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:
- on(a,b)
- on(b,c)
- ontable(c)



A plan:
- pickup(a)
- stack(a,b)
- unstack(a,b)
- putdown(a)
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

Note: Goals in a different order!

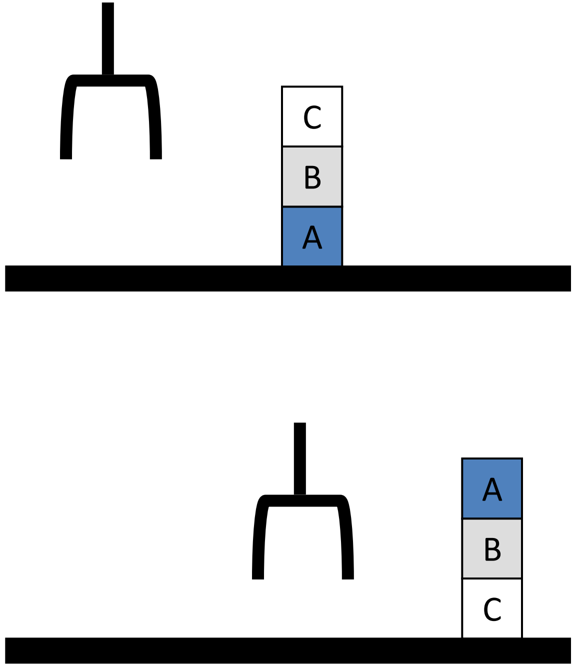
Typical BW planning problem

Initial state:

clear(c)
ontable(a)
on(b,a)
on(c,b)
handempty

Goal:

on(a,b)
on(b,c)
ontable(c)



Plan:

unstack(c,b)
putdown(c)
unstack(b,a)
putdown(b)
pickup(a)
stack(a,b)
unstack(a,b)
putdown(a)
pickup(b)
stack(b,c)
pickup(a)
stack(a,b)

Note: not very efficient!

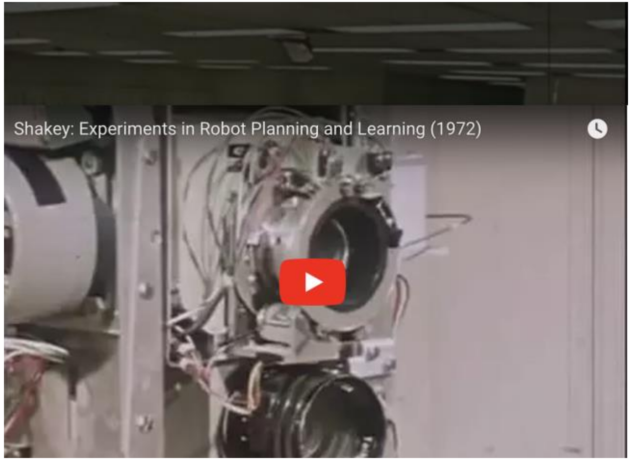
Major approaches

- Planning as search
- GPS / STRIPS
- Situation calculus
- Partial order planning
- Hierarchical decomposition (HTN planning)
- Planning with constraints (SATplan, Graphplan)
- Reactive planning

Shakey the robot

First general-purpose mobile robot to be able to reason about its own actions

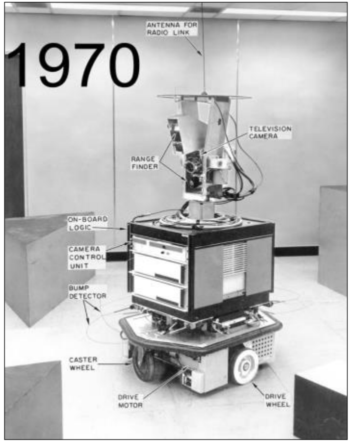
Shakey the Robot: 1st Robot to Embody Artificial Intelli- gence (2017, 6 min.)



Shakey: Experiments in Robot Planning and Learning (1972, 24 min)

Strips planning representation

- Classic approach first used in the STRIPS (Stanford Research Institute Problem Solver) planner
- A State is a conjunction of ground literals
  at(Home) $\land$ $\neg$have(Milk) $\land$ $\neg$have(bananas) ...
- Goals are conjunctions of literals, but may have variables, assumed to be existentially quantified
  at(?x) $\land$ have(Milk) $\land$ have(bananas) ...
- Need not fully specify state
  - Non-specified conditions either don't-care or assumed false
  - Represent many cases in small storage
  - May only represent changes in state rather than entire situation
- Unlike theorem prover, not seeking whether goal is true, but is there a sequence of actions to attain it



Shakey the robot

UMBC

Blocks world operators

- Classic basic operations for the blocks world
  - stack(X,Y): put block X on block Y
  - unstack(X,Y): remove block X from block Y
  - pickup(X): pickup block X
  - putdown(X): put block X on the table
- Each represented by
  - list of preconditions
  - list of new facts to be added (add-effects)
  - list of facts to be removed (delete-effects)
  - optionally, set of (simple) variable constraints
- For example stack(X,Y):

preconditions(stack(X,Y), [holding(X), clear(Y)])
deletes(stack(X,Y), [holding(X), clear(Y)]).
adds(stack(X,Y), [handempty, on(X,Y), clear(X)])
constraints(stack(X,Y), [X≠Y, Y≠table, X≠table])

STRIPS planning

- STRIPS maintains two additional data structures:
  - State List - all currently true predicates.
  - Goal Stack - push down stack of goals to be solved, with current goal on top
- If current goal not satisfied by present state, find action that adds it and push action and its preconditions (subgoals) on stack
- When a current goal is satisfied, POP from stack
- When an action is on top stack, record its application on plan sequence and use its add and delete lists to update current state

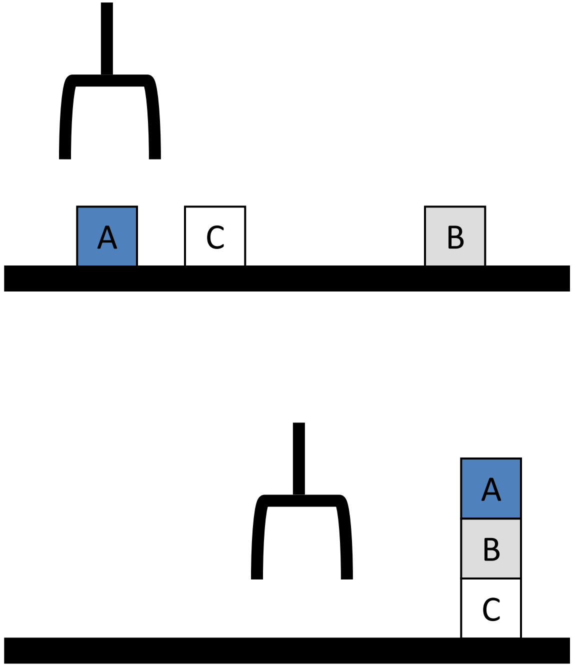
Typical BW planning problem

Initial state:

- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:

- on(b,c)
- on(a,b)
- ontable(c)



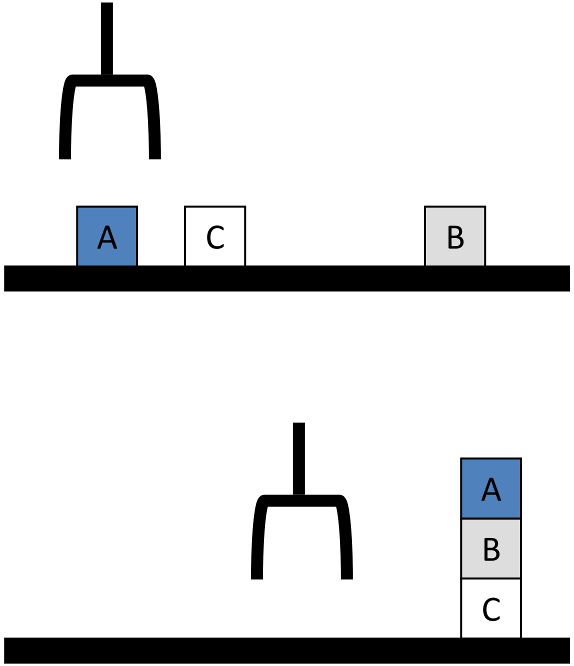
A plan:

- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

Typical BW planning problem

Initial state:
- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:
- on(a,b)
- on(b,c)
- ontable(c)



A plan:
- pickup(a)
- stack(a,b)
- unstack(a,b)
- putdown(a)
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

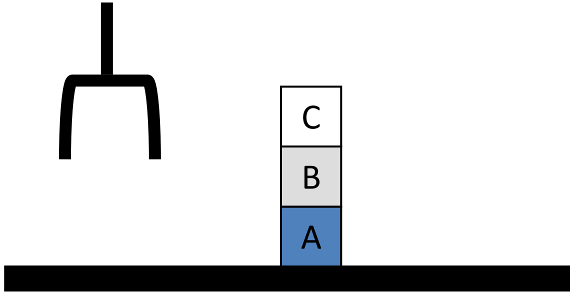
Typical BW planning problem

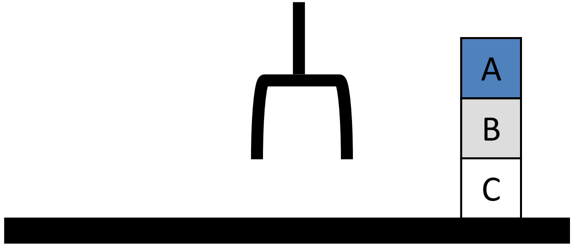
Initial state:

clear(c)
ontable(a)
on(b,a)
on(c,b)
handempty

Goal:

on(a,b)
on(b,c)
ontable(c)





Plan:

unstack(c,b)
putdown(c)
unstack(b,a)
putdown(b)
pickup(b)
stack(b,a)
unstack(b,a)
putdown(b)
pickup(a)
stack(a,b)
unstack(a,b)
putdown(a)
pickup(b)
stack(b,c)
pickup(a)
stack(a,b)

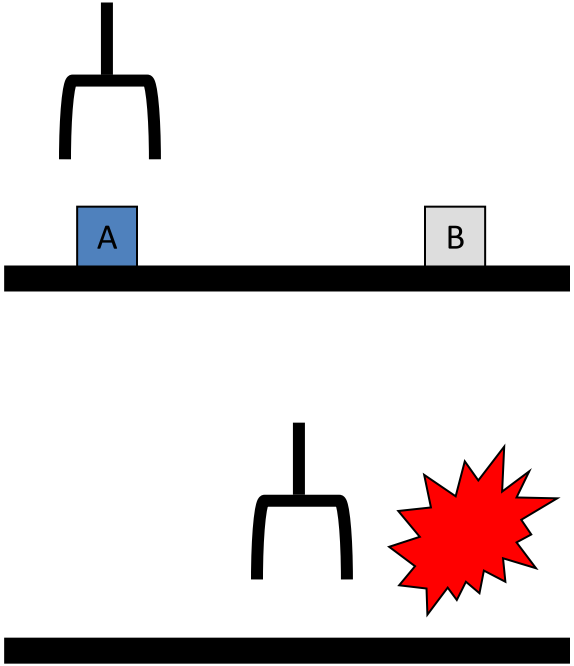
Typical BW planning problem

Initial state:

- ontable(a)
- ontable(b)
- clear(a)
- clear(b)
- handempty

Goal:

- on(a,b)
- on(b,a)

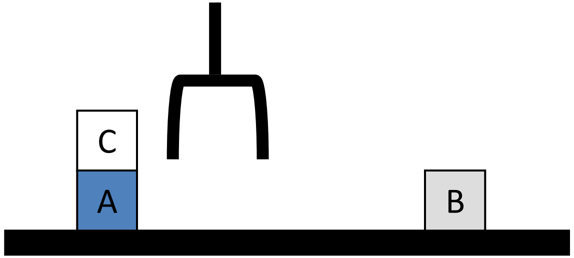


Plan:

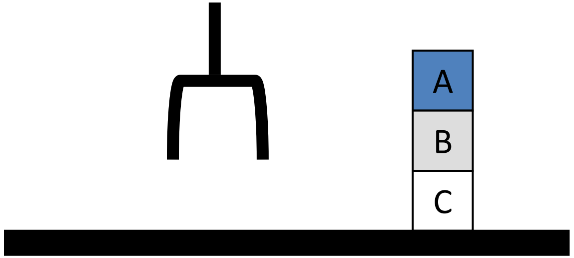
??

Goal interaction

- Simple planning algorithms assume independent sub-goals
  - Solve each separately and concatenate the solutions
- The "Sussman Anomaly" is the classic example of the goal interaction problem:
  - Solving on(A,B) first (via unstack(C,A), stack(A,B)) is undone when solving 2nd goal on(B,C) (via unstack(A,B), stack(B,C))
  - Solving on(B,C) first will be undone when solving on(A,B)
- Classic STRIPS couldn't handle this, although minor modifications can get it to do simple cases

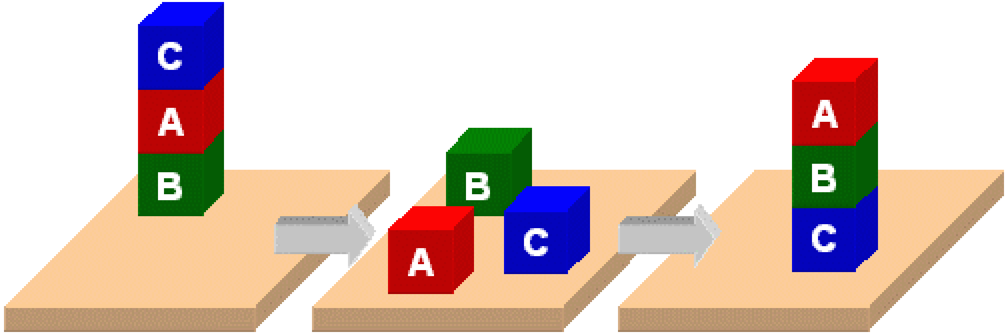


Initial state

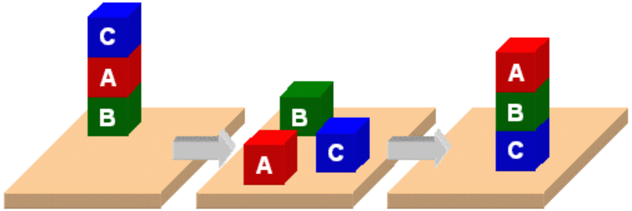


Goal state

PDDL



PDDL



- **Planning Domain Description Language**
- Based on STRIPS with various extensions
- First defined by Drew McDermott (Yale) et al.
  - Classic spec: PDDL 1.2; good reference guide
- Used in biennial International Planning Competition (IPC) series (1998-2020)
- Many planners use it as a standard input

PDDL Representation

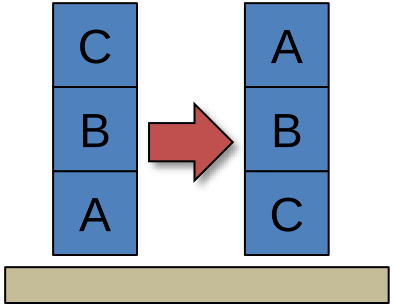
- Task specified via two files: **domain file** and **problem file**
  - Both use a logic-oriented notation with Lisp syntax
- **Domain file** defines a domain via *requirements*, *predicates*, *constants*, and *actions*
  - Used for many different problem files
- **Problem file:** defines problem by describing its *domain*, *objects*, *initial state* and *goal state*
- **Planner:** takes a domain and a problem and produces a plan

Blocks Word Domain File


```
(define (domain BW)
  (:requirements :strips)
  (:constants red green blue yellow small large)
  (:predicates (on ?x ?y) (on-table ?x) (color  ?x ?y) … (clear
      ?x))
  (:action pick-up
     :parameters (?obj1)
     :precondition (and (clear ?obj1) (on-table ?obj1)
                                    (arm-empty))
     :effect (and (not (on-table ?obj1))
                        (not (clear ?obj1))
                        (not (arm-empty))
                        (holding ?obj1)))
  … more actions ...)
```

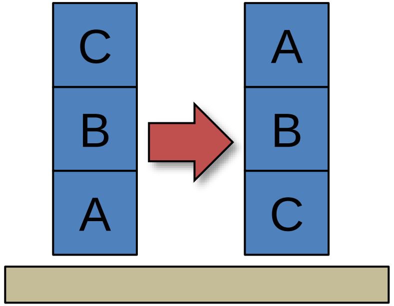
Blocks Word Problem File

```
(define (problem 00)
   (:domain BW)
   (:objects A B C)
   (:init (arm-empty)
             (on B A)
             (on C B)
             (clear C))
   (:goal (and (on A B)
                  (on B C))))
```



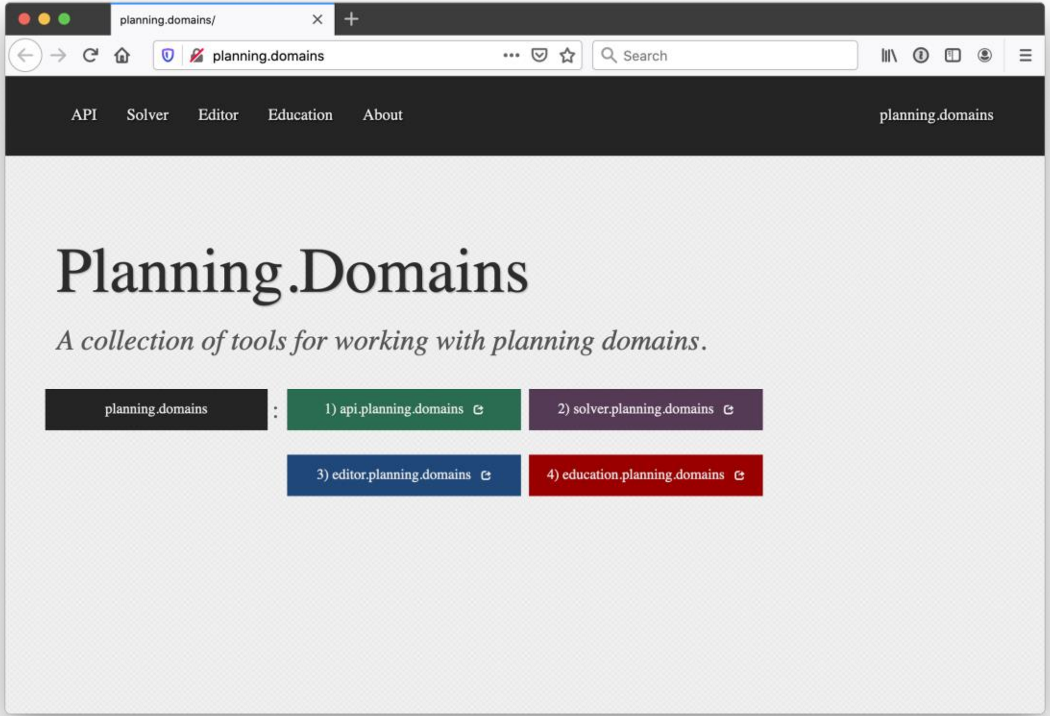
Blocks Word Problem File

```
(define (problem 00)
   (:domain BW)
   (:objects A B C)
   (:init (arm-empty)
             (on B A)
             (on C B)
             (clear C))
   (:goal (and (on A B)
                 (on B C))))
```



Begin plan
1 (unstack c b)
2 (put-down c)
3 (unstack b a)
4 (stack b c)
5 (pick-up a)
6 (stack a b)
End plan

http://planning.domains/



Planning.domains

- Open source environment for providing planning services using PDDL (GitHub)
- Default planner is ff
  - very successful forward-chaining heuristic search planner producing sequential plans
  - Can be configured to work with other planners
- Use interactively or call via web-based API