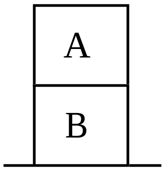
An Opening Exercise

Assume that a robot is given this set of operators:

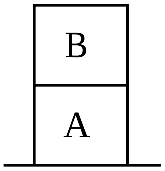
```
stack( x, y )

   precondition: clear( y ), holding( x )
   add:          armEmpty, on( x, y )
   delete:       clear( y ), holding( x )

unstack( x, y )

   precondition: on( x, y ), clear( x ), armEmpty
   add:          holding( x ), clear( y )
   delete:       on( x, y ), armEmpty

pickup( x )

   precondition: clear( x ), on( x, TABLE ), armEmpty
   add:          holding( x )
   delete:       on( x, TABLE ), armEmpty

putdown( x )

   precondition: holding( x )
   add:          on( x, TABLE ), armEmpty
   delete:       holding( x )
```

Solve:

Initial state:

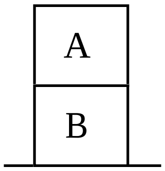


Goal state:

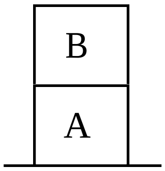


A Solution to the Exercise

Initial state:



Goal state:



```
I = clear( A )          G = on( B, A )
    on( A, B )
    on( B, TABLE )
    armEmpty
```

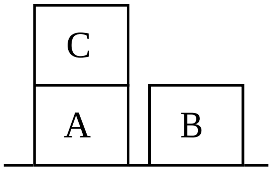
Another Problem To Solve

Assume that a robot is given the same set of operators:

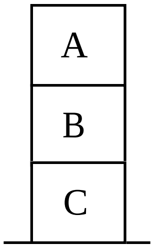
```
stack( x, y )

  precondition: clear( y ), holding( x )
  add:          armEmpty, on( x, y )
  delete:       clear( y ), holding( x )

unstack( x, y )

  precondition: on( x, y ), clear( x ), armEmpty
  add:          holding( x ), clear( y )
  delete:       on( x, y ), armEmpty

pickup( x )

  precondition: clear( x ), on( x, TABLE ), armEmpty
  add:          holding( x )
  delete:       on( x, TABLE ), armEmpty

putdown( x )

  precondition: holding( x )
  add:          on( x, TABLE ), armEmpty
  delete:       holding( x )
```

Solve:

Initial state:

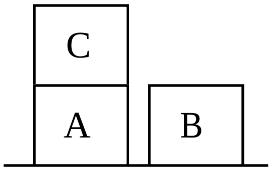


Goal state:

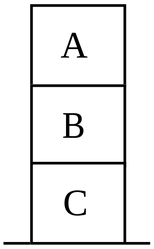


A Problem with the Solution

Initial state:



Goal state:



Our **goal-stack planning algorithm** fails, even though a straightforward plan exists:

```
unstack( C, A )
putdown( C )
pickup( B )
stack( B, C )
pickup( A )
stack( A, B )
```

Why can't our algorithm find this plan?

It achieves each subgoal before considering any other.

But:

- If it tries to achieve `stack( B, C )` first...
- If it tries to achieve `stack( A, B )` first...

Features of Goal-Stack Planning

This algorithm...

- ... is non-deterministic
- ... is backward chaining
- ... generates a **totally-ordered plan**.
- ... uses a "most commitment" strategy.
- ... is **non-interleaving**.

  work on `on(A,B)`
  work on `on(B,C)`
  work on `on(A,B)`

- ... does search based on the state of the world.
- ... is **incomplete**.

Performance of the Algorithm Against The Key Ideas of Planning

1. Planning problems are decomposable and (mostly) independent, so our planner should be able to recognize this and use it to the planner’s advantage.

2. So, states should be decomposable.

3. A planner should be able to choose any action that makes sense and add it to the “right place” in the plan at any time.

Planning as Search

Using goal-stack planning, an agent searches through a space of *world states*.

- What is the state of the world after applying a set of operators?
- How can we move to a better state?

This is one of the problems with goal-stack planning.

It seems that an intelligent planners think about goals, how to achieve them, how to de-compose them, and how to reassemble sub-plans into a plan that achieves the initial goals.

A goal-stack planner thinks about **world states**.

Intelligent planners seem to think about **plans**.

Redefining the Search Problem

Why can’t our agent *reason about* ***plans***?

We need an algorithm that searches over “states” that are (partially completed) plans.

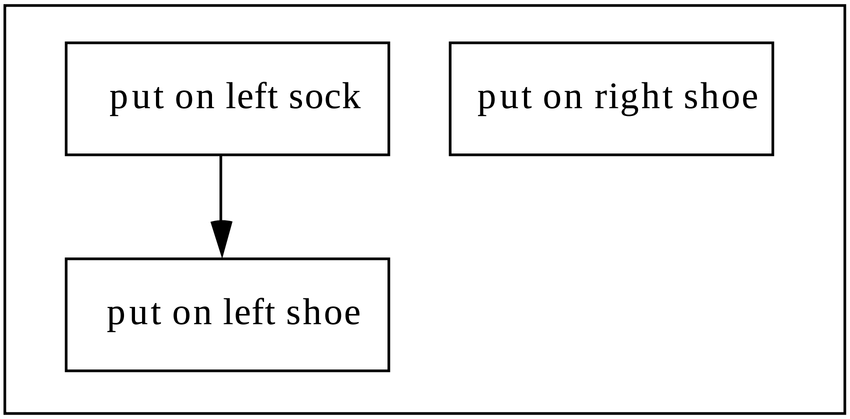
In **plan-space planning**, we search through a space of *plans* that might serve our purpose.

This is a common approach in computing: When we cannot find a suitable solution to a problem, **change levels of abstraction** and try to solve it there.

(Another common approach is to **add a level of indirection**.)

Plan-Space Planning

A **plan** is a **set of steps** and a **set of constraints** on the ordering of the steps.



In our search, a **state** is a plan.

- The plan may be incomplete, or *partial*. A partial plan may be missing a necessary step or a necessary constraint on their ordering.

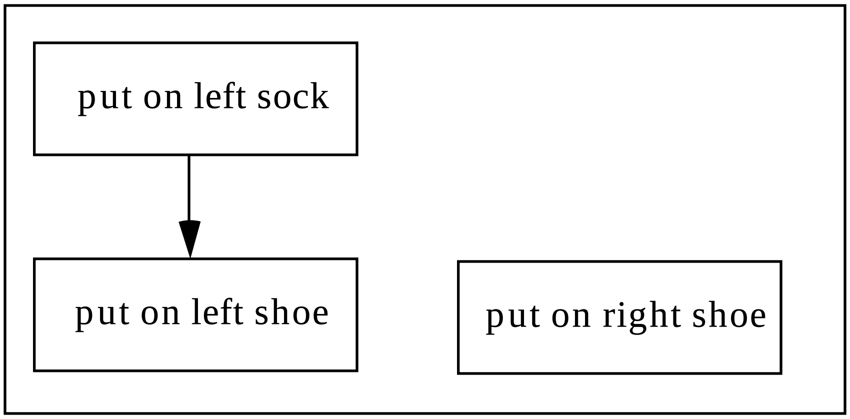
- A *complete* plan has all the steps it needs and all of the contraints on the ordering of the steps it needs.

The **goal state** is a complete plan that achieves all of the required goals.

Operators in Plan Space

If states are plans, then...

an **operator** creates **a new plan from an old plan**.



What could we do to this plan to create a different plan?