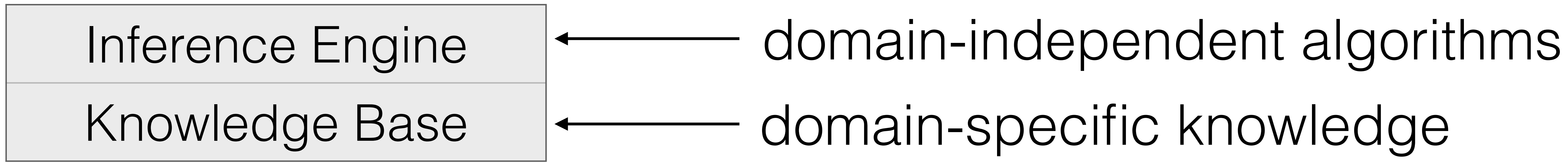
COMP813 Artificial Intelligence

Module 7: Knowledge Representation in Planning and General Game Playing (1)

Auckland University of Technology

“Humans, it seems, know things; and what they know helps them do things. These are not empty statements. They make strong claims about how the intelligence of humans is achieved—not by purely reflex mechanisms but by processes of **reasoning** that operate on internal **representations** of knowledge. In AI, this approach to intelligence is embodied in **knowledge-based agents**.”

Logical Agents



- **Knowledge Base KB** = set of sentences in a *formal language*
- **Declarative approach** to building an agent:
  - **Tell** agent what it needs to know (instead of how to do)
  - **Ask** itself what to do (answers should follow from KB)

Imperative vs Declarative

Dinning at a restaurant

- An **imperative** approach (**HOW**): "*I see that table located in the middle is empty. My friend and I are going to walk over there and sit down.*"

- A **declarative** approach (**WHAT**): "*Table for two, please.*"

Imperative vs Declarative

Programming languages:

- Imperative: C, C++, Java
- Declarative: SQL, HTML, GDL

```sql
SELECT * FROM Users WHERE Country='Mexico';
```

```html
<article>
  <header>
    <h1>Declarative Programming</h1>
    <p>Sprinkle Declarative in your verbiage to sound smart</p>
  </header>
</article>
```

A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```

The agent must be able to:

- Represent **states, actions**, and incorporate **new percepts**
- **Update internal representations** of the world
- **Deduce hidden properties** of the world
- **Deduce appropriate actions**

Wumpus World

Performance measure

- gold +1000, death -1000
- -1 per step, -10 for using the arrow

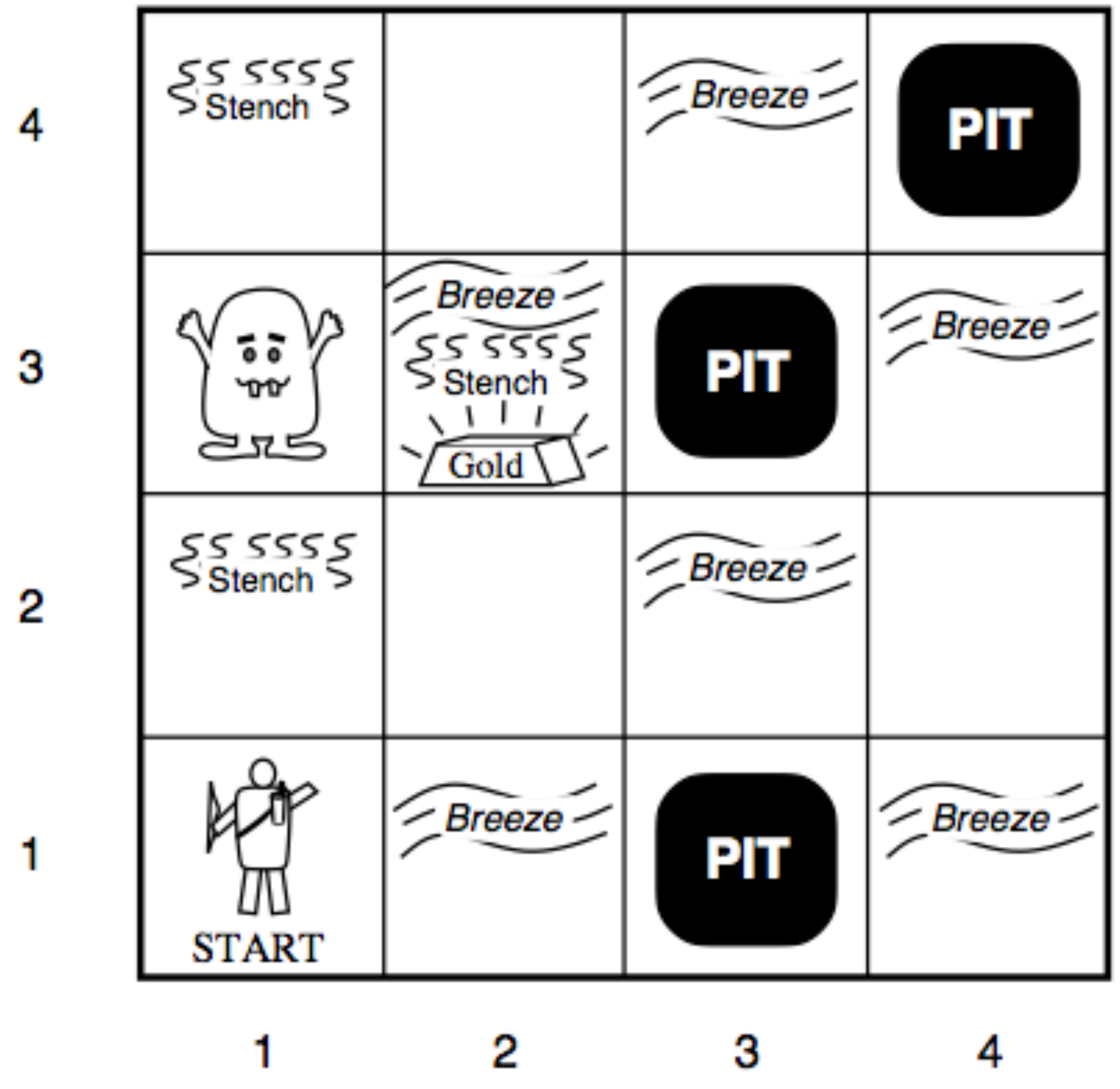
Environment

- Squares adjacent to wumpus are ***smelly***
- Squares adjacent to pit are ***breezy***
- ***Glitter*** if gold is in the same square
- ***Shooting*** kills wumpus if the agent is facing it
- ***Shooting*** uses up the only arrow
- ***Grabbing*** picks up gold if in same square
- ***Releasing*** drops the gold in same square

Actuators Left turn, Right turn, Forward, Grab, Release, Shoot

Sensors Breeze, Glitter, Smell (*Incomplete information*: the agent can only sense the current square)

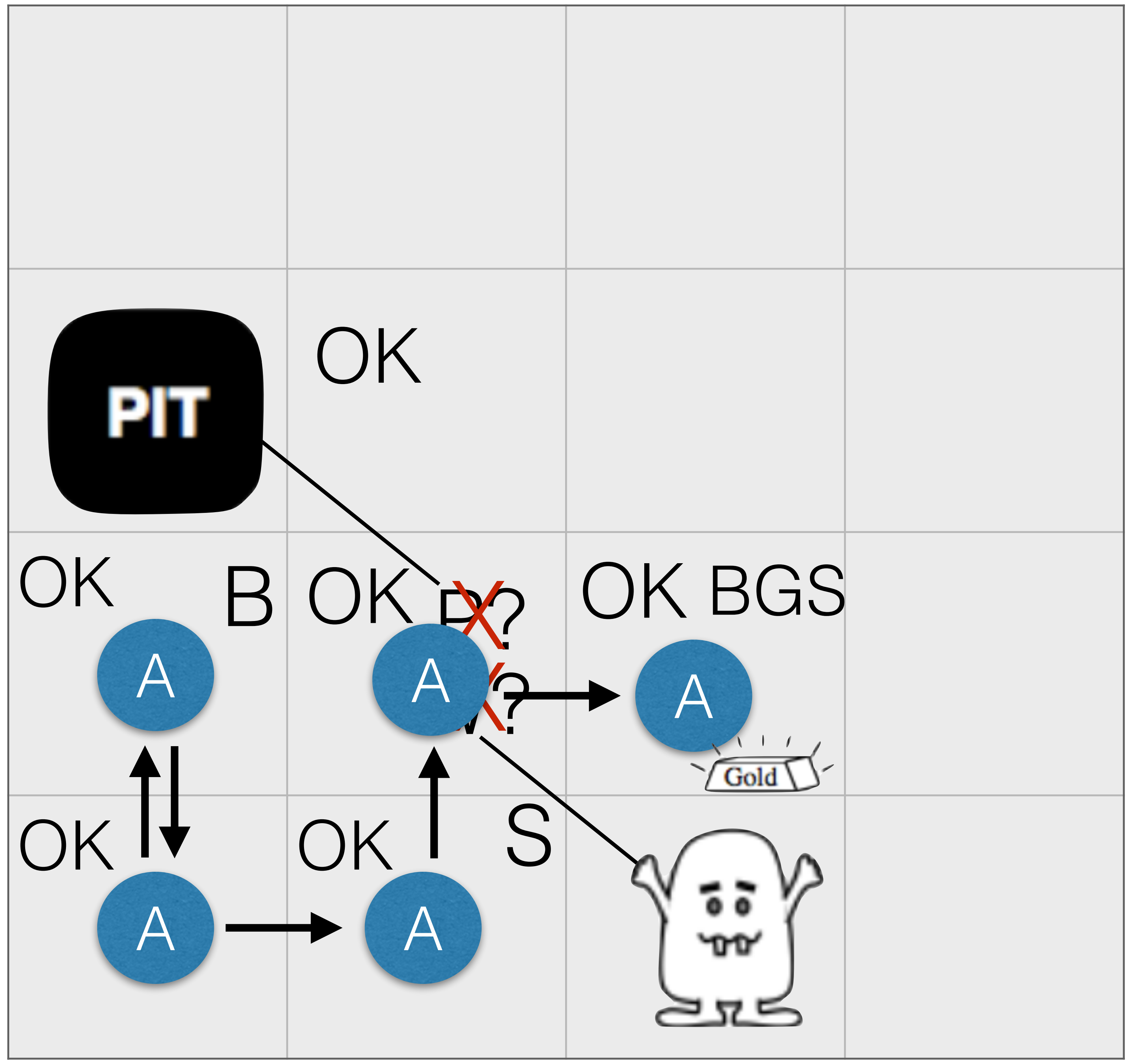
Example from AIMA



An example from the AIMA book (see the reference)

Wumpus World

A: Agent
OK: Square is safe
B: Breeze
P: Pit
S: Smell
W: Wumpus
G: Glitter



Logic for Knowledge Base

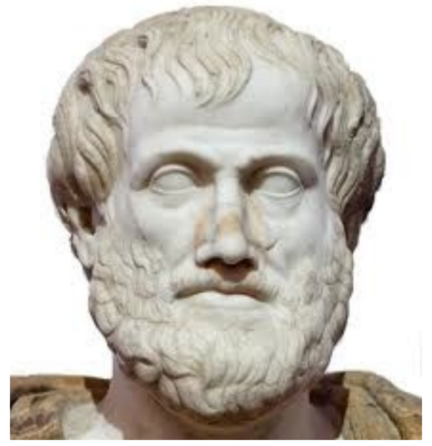
Knowledge Base uses logic for representing information such that conclusions can be drawn.

Syntax	the sentences in the language
Semantics	the “meaning” of sentences

Logic Reasoning

Socrates is a man
All men are humans
All humans are mortal

Socrates is a man
All men are humans → Socrates is a human



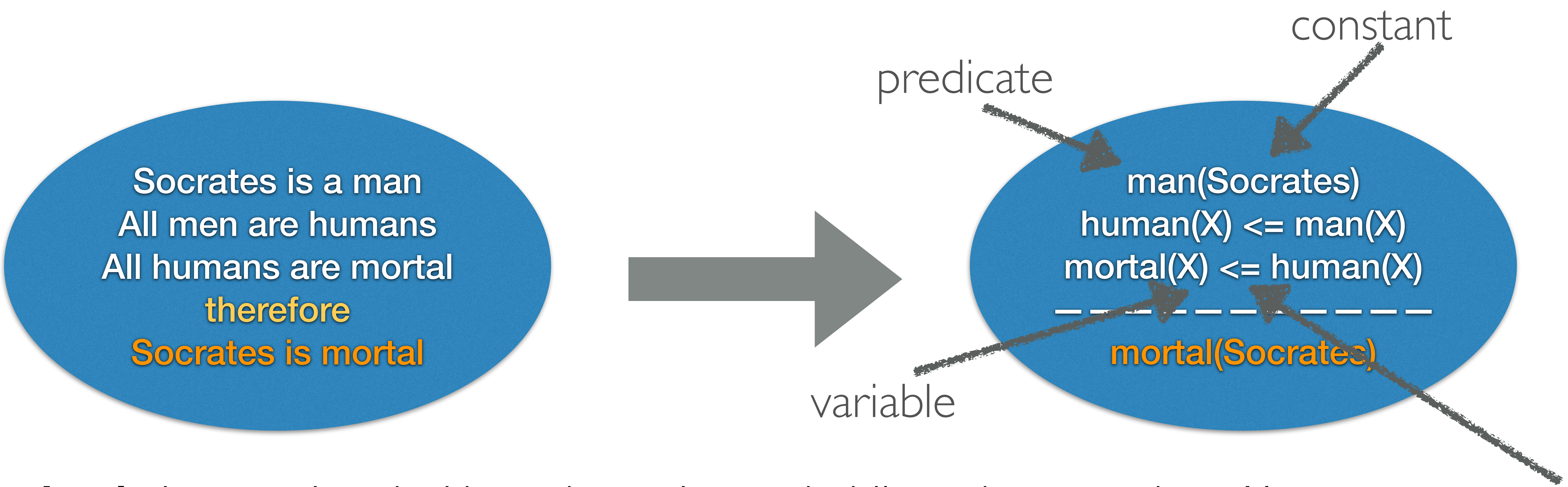
Aristotle:
Syllogisms.

Socrates is a man
All men are humans
All humans are mortal
Socrates is a human

Socrates is a man
All men are humans
All humans are mortal
Socrates is human
Socrates is mortal

Socrates is a human
All humans are mortal → Socrates is mortal

Logic Reasoning



- **Logic** is associated with mathematics and philosophy to modern AI
- As one of the main formalisms for knowledge representation and automated reasoning in AI
- PROLOG (PROgramming in LOGic) is a logic programming language for AI.
- **Game Description Language (GDL)** is a logic-based language for describing game for General Game Playing. (GDL is similar to PROLOG).

A First-Order Logic (FOL) fragment

Basic symbols in FOL:

Quantifier	∀ (for all) ∃ (exist)
Constants	`john, alice, apple, xplayer, oplayer`
Variables	`X, Y, Z`
Predicates	`p, q, human(), mortal(), cell( , , )`
Functions	`mother(), mark( , )`
Logic connectives	¬ (not), /\ (and), \/ (or), <= (reverse implication)

the **arity** of functions and predicates is the number of arguments they can take.
e.g., `p` has arity 0, `human()` has arity 1, `cell( , , )` has arity 3

Term:

a variable	`X`
a constant	`john`
or a function term	`mother(john)`

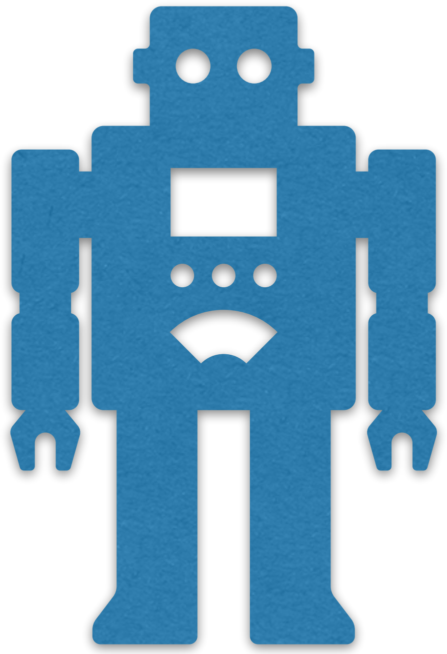
Sentence:

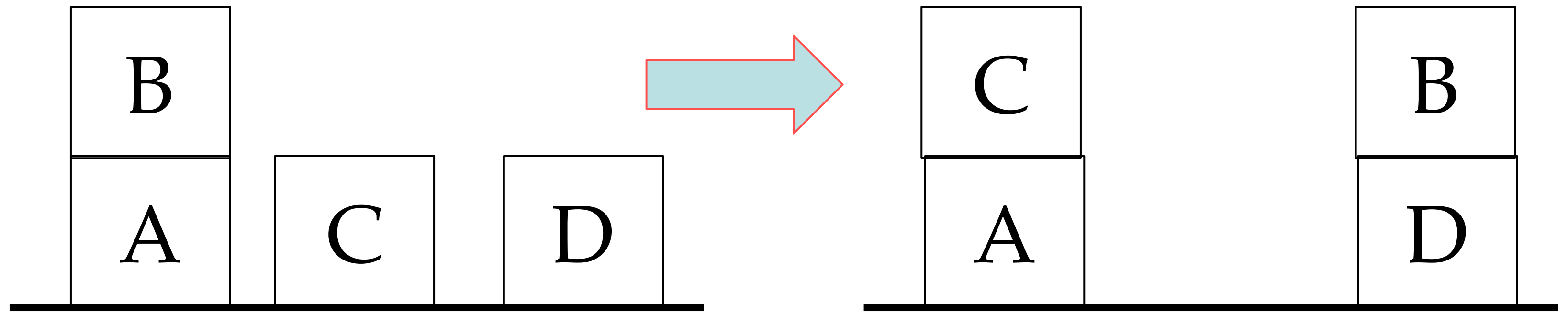
an atomic sentence (a proposition)	`human(mother(John))`
or a complex sentence with variables	`∀X (mortal(X) <= human(X))`

A proposition (i.e., atomic sentence) is either true or false in any situations (states)

Planning

A Robotic Planning Problem

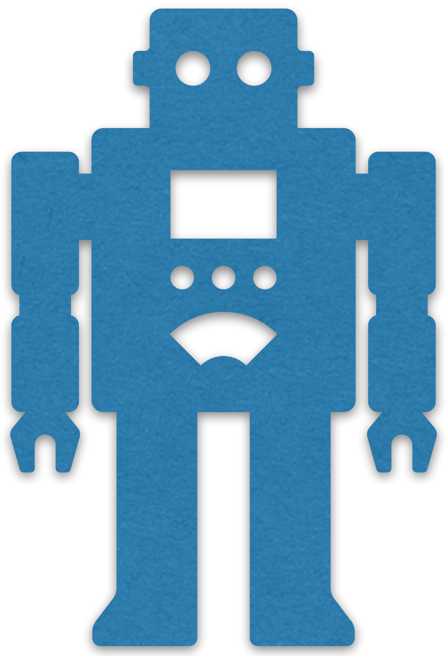


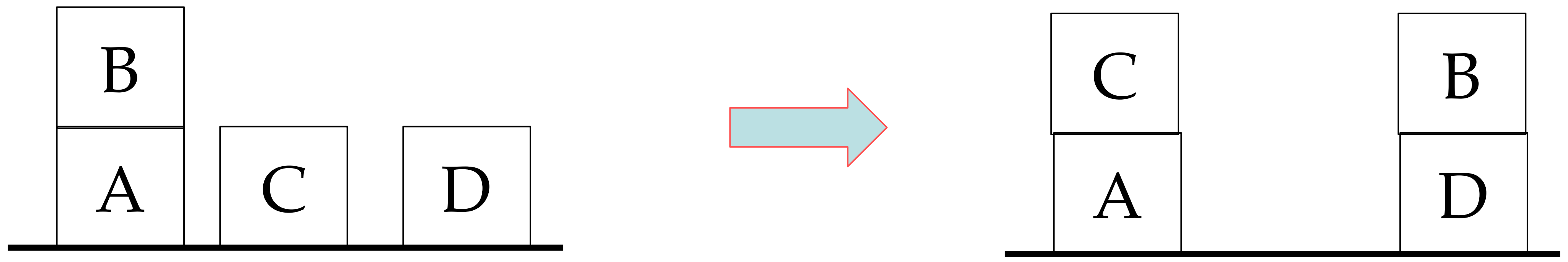


Start Goal

How do we **represent** the states of the world and the **rules** needed?

Representing the World





ON(x,y): Block x is on Block y
CLEAR(x): Top of Block x is clear.
ONTABLE(x): Block x is on table.
ARMEMPTY: Robotic arms are empty

Start

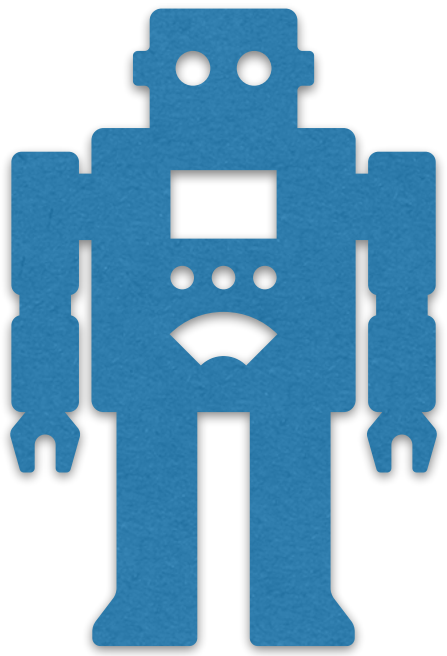
ON(B,A) & CLEAR(B) &
CLEAR(C) & CLEAR(D) &
ONTABLE(A) & ONTABLE(C) &
ONTABLE(D) & ARMEMPTY

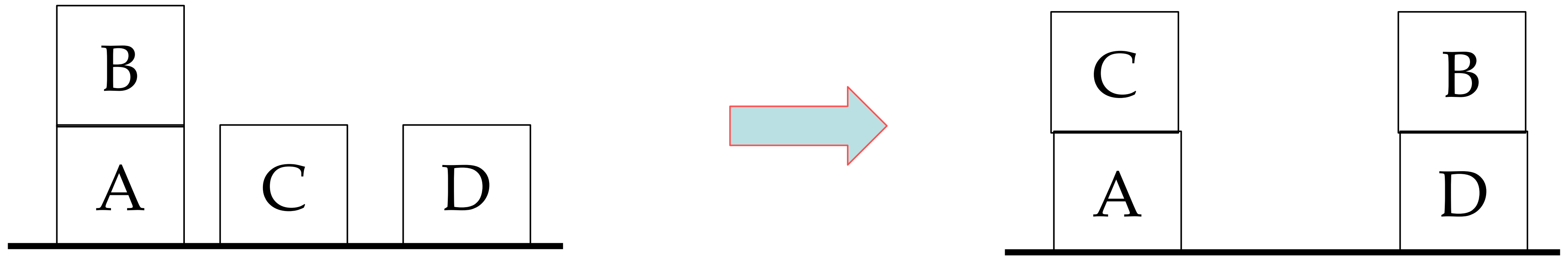
Goal

ON(C,A) &
ON(B,D) &
ONTABLE(A) &
ONTABLE(D)

How about
Adjacent(A,C) & Adjacent(C,D),
Near-the-edge(D),
In-between(A,C,D), etc.

Representing the actions



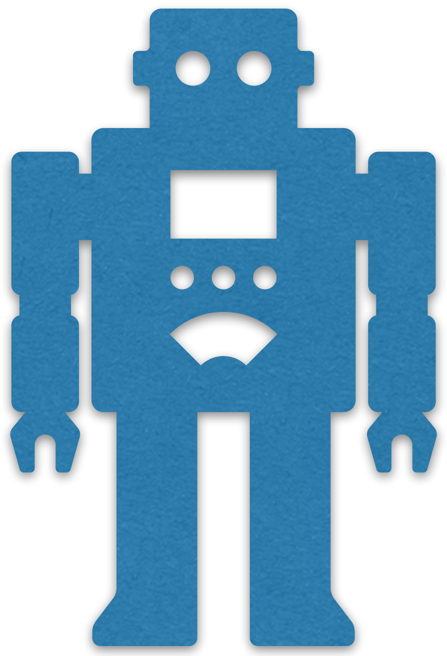


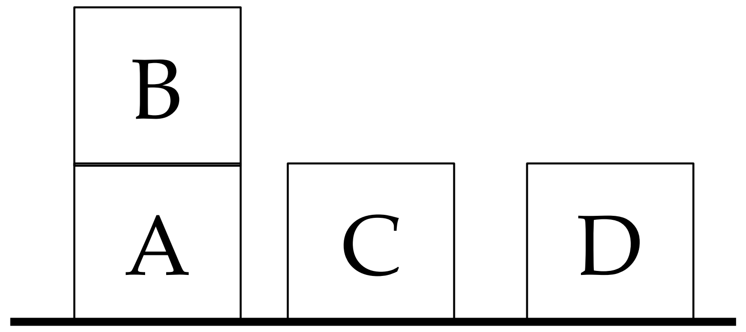
Question: what kind of operators do I need/have? i.e what actions?

STACK, UNSTACK, PICKUP, PUTDOWN

How do we represent them? Or, what would a function for Stack be like?

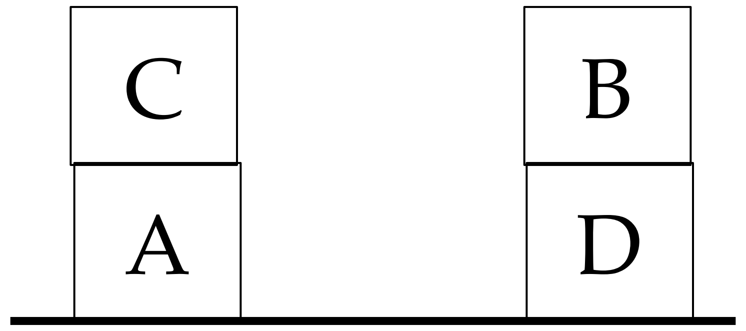
Representing the actions





Start

ON(B,A) & CLEAR(B) & CLEAR(C) & CLEAR(D) & ONTABLE(A) & ONTABLE(C) & ONTABLE(D) & ARMEMPTY



Goal

ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

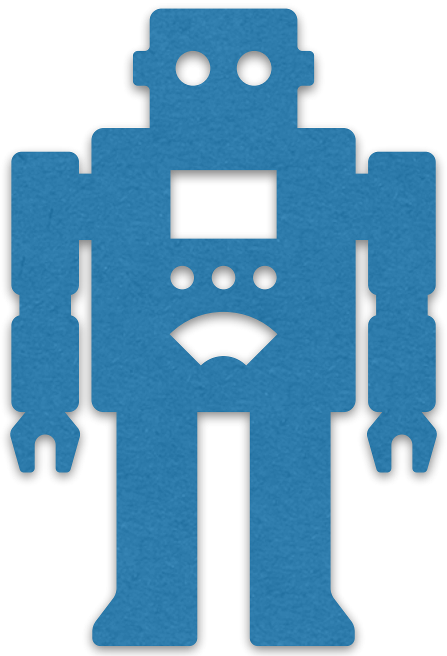
How to represent **Stack(C,A)**?

Need to hold C?

Need to make sure A is clear?

If the above two are true, what does it mean to carry out the action, stacking C on A?

Representing the actions



Need to hold C?

Need to make sure A is clear?

Some pre-requisites

If the above two are true, what does it mean to carry out the action, stacking C on A?

The world changes its states

STRIPS-Style Representation

- **STRIPS (Stanford Research Institute Problem Solver)** is an **automated planning system** in AI.
- Each action in a STRIPS-style representation consists of
  - **A preconditions**, which specify the conditions that must be true before the action can be taken, and
  - **an effect**, which specify the changes that the action will make to the state of the world.
- Applied to a wide range of problem domains beyond the original block world domain, including **scheduling, robotics, and transportation planning.**

STRIPS-Style Representation

STRIPS-style **PDA** operations for the block world

STACK(x,y)

P: CLEAR(y) & HOLDING(x)
D: CLEAR(y) & HOLDING(x)
A: ARMEMPTY & ON(x,y)

UNSTACK(x,y)

P: ON(x,y) & ARMEMPTY
D: ON(x,y) & CLEAR(x) & ARMEMPTY
A: HOLDING(x) & CLEAR(y)

P: pre-conditions; D: delete; A: add

STRIPS-Style Representation

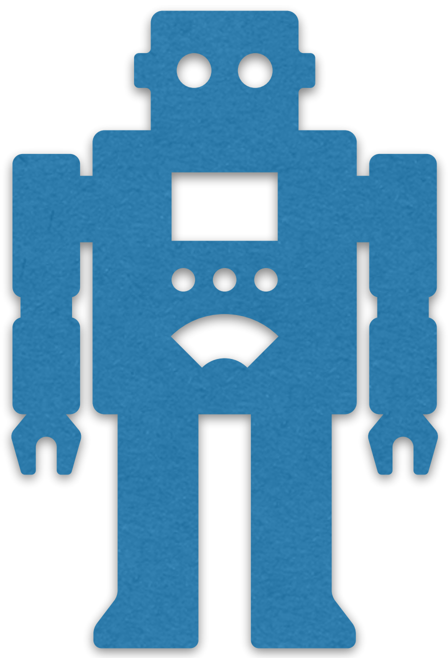
STRIPS-style **PDA** operations for the block world

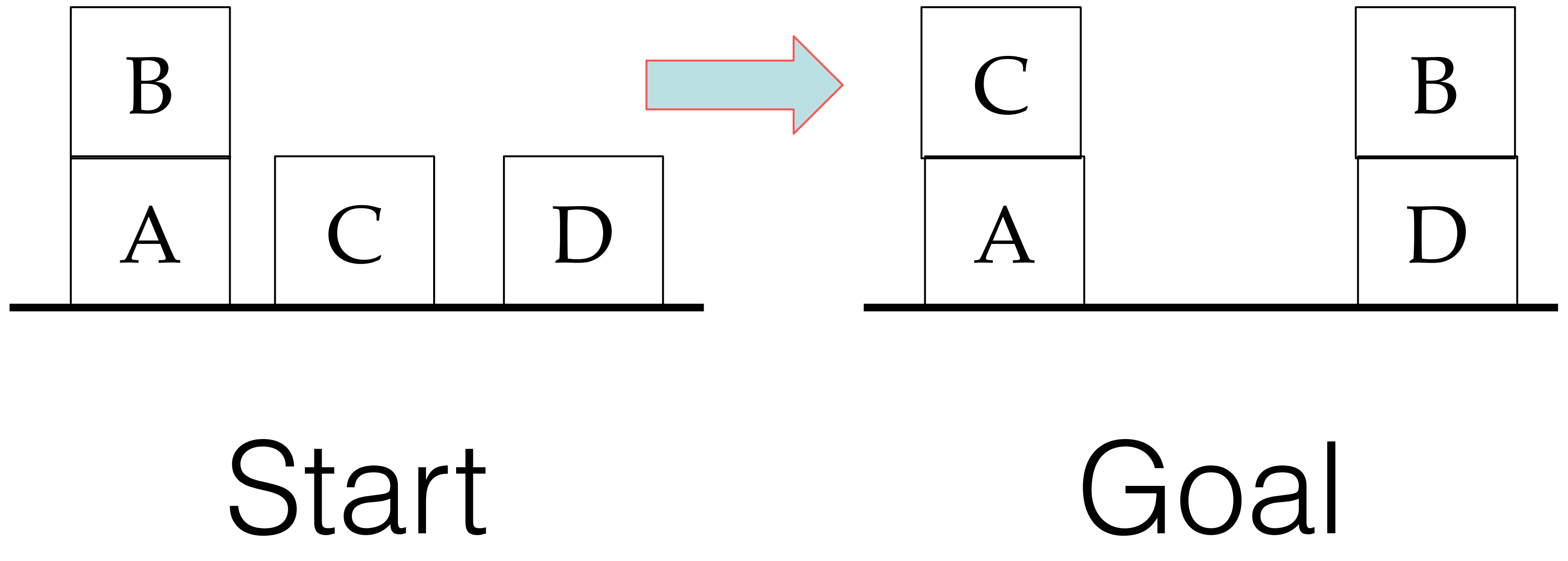
Exercise: Write PICKUP and PUTDOWN.

PICKUP(x)
P: CLEAR(x) & ONTABLE(x) & ARMEMPTY
D: ONTABLE(x) & ARMEMPTY
A: HOLDING(x)

PUTDOWN(x)
P: HOLDING(x)
D: HOLDING(x)
A: ONTABLE(x) & ARMEMPTY

How to find a plan?



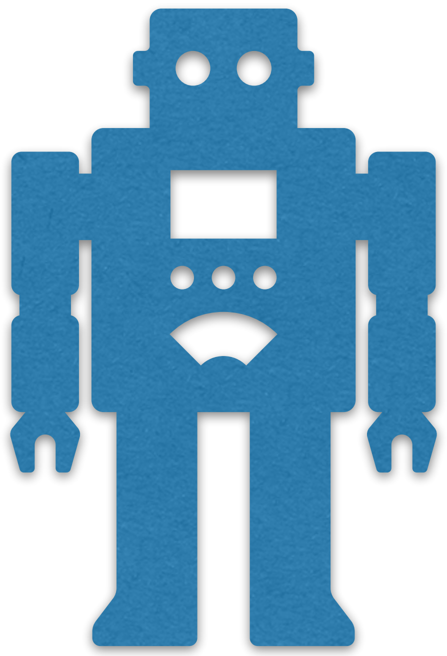


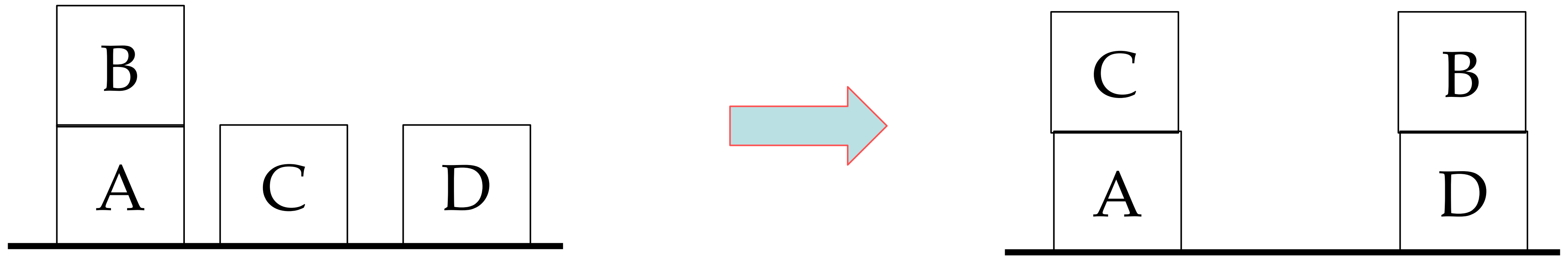
STACK, UNSTACK, PICKUP, PUTDOWN

YOU, looking at the GOAL states, can decide: **UNSTACK(B), STACK(B,D), PICKUP(C)** and **STACK(C,A)**, right?

But how does the robot know?!

How to find a plan?





Start

ON(B,A) & CLEAR(B) & CLEAR(C) & CLEAR(D) & ONTABLE(A) & ONTABLE(C) & ONTABLE(D) & ARMEMPTY

Goal

ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

To form a plan

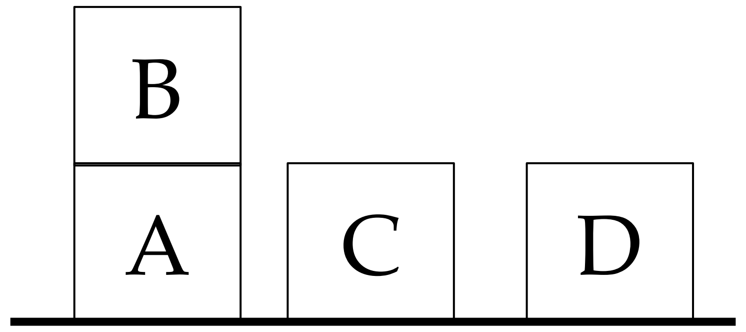
1. Input a list of goals, G, to be achieved
2. Unless input list is empty, pick one goal **(but you have multiple goals!)**
3. Find a suitable action or a set of actions that will enable us to accomplish that goal **(but which ones?)**
4. Execute actions, if successful repeat step 2 **(but how do you choose which actions to execute first?)**.
5. Repeat Step 3.

Knowledge Base + Inference

Planning

First Step - Create a goal stack:

Start



ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

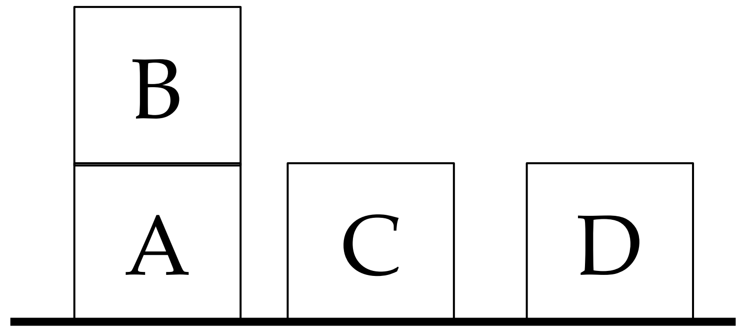
GOAL: ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

A stack is a Data Structure: an ordered collection of items where the addition of new items and the removal of existing items always takes place at the same end.

Planning

Second Step – Check which sub-goals are not satisfied and create them as new goals on the stack.

Start



Note that ONTABLE(A) & ONTABLE(D) are satisfied in Start state.

GOAL: ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

ON(B,D)

ON(C,A)

ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

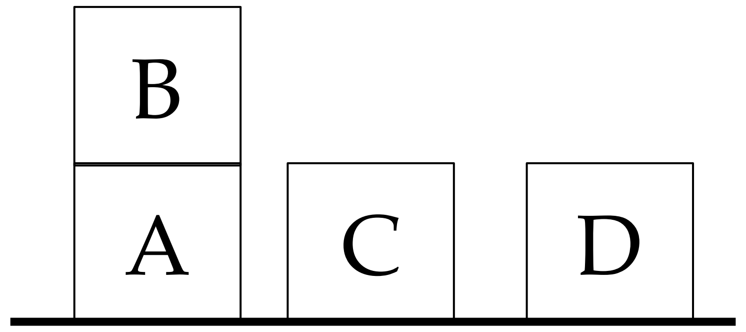
Goal stack

Goal stack is updated with 2 new subgoals. Is this the only possible new Goal stack?

Planning

Second Step – Check which sub-goals are not satisfied and create them as new goals on the stack.

Start



GOAL: ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

ON(B,D)

ON(C,A)

ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

Goal stack

Possibility 1

ON(C,A)

ON(B,D)

ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

Goal stack

Possibility 2

Which one to choose?

Simple strategy: leftmost or rightmost first.

Planning

Next Step – What do we do now?

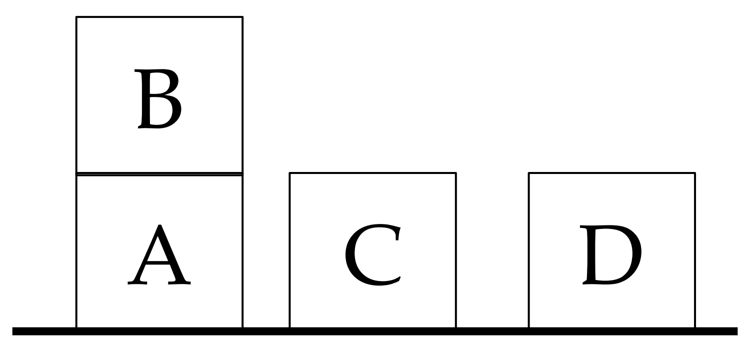
ON(C,A)

ON(B,D)

ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

Start

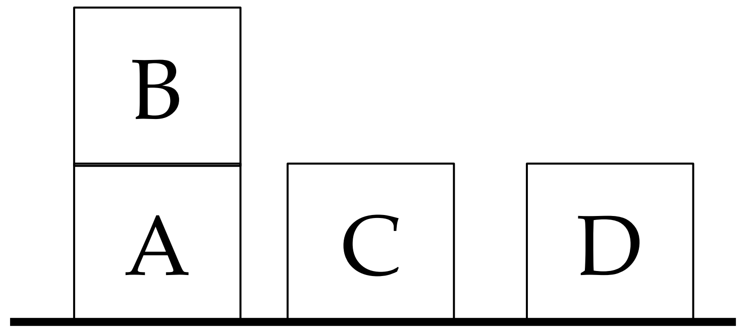


GOAL: ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

We try to satisfy ON(C,A) by selecting the appropriate operator(s) to do so.

Planning

Start



Next Step – What do we do now?

ON(C,A)

ON(B,D)

ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

Which operator is useful for ON(C,A)?

STACK(*x*,*y*)
P: CLEAR(*y*) & HOLDING(*x*)
D: CLEAR(*y*) & HOLDING(*x*)
A: ARMEMPTY & ON(*x*,*y*)

PICKUP(*x*)
P: CLEAR(*x*)&ONTABLE(*x*)&
ARMEMPTY
D: ONTABLE(*x*) & ARMEMPTY
A: HOLDING(*x*)

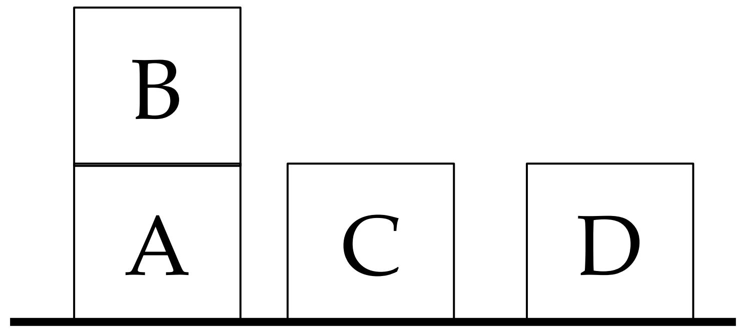
UNSTACK(*x*,*y*)
P: ON(*x*,*y*) & CLEAR(*x*) &
ARMEMPTY
D: ON(*x*,*y*) & ARMEMPTY
A: HOLDING(*x*) & CLEAR(*y*)

PUTDOWN(*x*)
P: HOLDING(*x*)
D: HOLDING(*x*)
A: ONTABLE(*x*) & ARMEMPTY

We try to satisfy ON(C,A) by selecting the appropriate operator(s) to do so.

Planning

Start



Next Step – We replace ON(C,A) with STACK(C,A):

~~ON(C,A)~~ STACK(C,A)

ON(B,D)

ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

Which operator is useful for ON(C,A)?

STACK(*x*,*y*)
P: CLEAR(*y*) & HOLDING(*x*)
D: CLEAR(*y*) & HOLDING(*x*)
A: ARMEMPTY & ON(*x*,*y*)

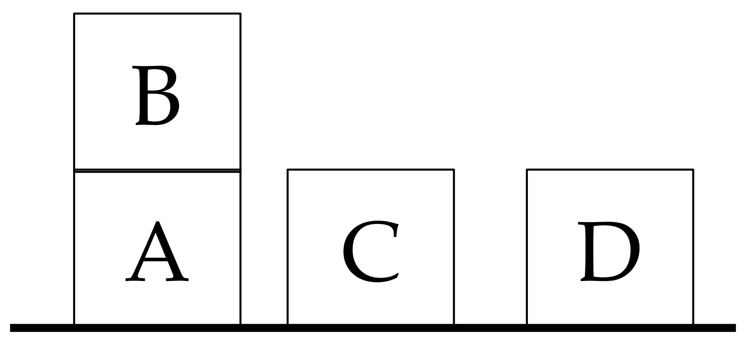
ON(x,y) and **ON(C,A)** are unified with a unifier:
(x=C, y=A)
This is then applied to the full operator rule.

STACK(*C*,*A*)
P: CLEAR(*A*) & HOLDING(*C*)
D: CLEAR(*A*) & HOLDING(*C*)
A: ARMEMPTY & ON(*C*,*A*)

We try to satisfy ON(C,A) by selecting the appropriate operator(s) to do so.

Planning

Start



Next Step – We replace ON(C,A) with STACK(C,A):

STACK(C,A)
ON(B,D)
ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

Which operator is useful for ON(C,A)?

STACK(*x,y*)
P: CLEAR(*y*) & HOLDING(*x*)
D: CLEAR(*y*) & HOLDING(*x*)
A: ARMEMPTY & ON(*x,y*)

ON(x,y) and **ON(C,A)** are unified with a unifier:
(x=C, y=A)
This is then applied to the full operator rule.

STACK(*C,A*)
P: CLEAR(*A*) & HOLDING(*C*)
D: CLEAR(*A*) & HOLDING(*C*)
A: ARMEMPTY & ON(*C,A*)

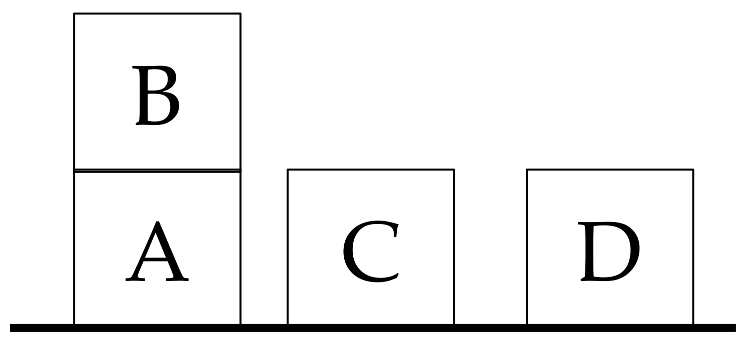
Here we take the first goal on the stack and if not satisfied, replaces it with an operator that when executed will create such a goal state. Why do we put the operator on the stack?

Because there is no guarantee that the operator can be operated upon immediately

How do we know if the operator can be operated upon? **Check if the pre-conditions of the operator are satisfied.**

Planning

Start



Next Step – Push the conditions onto the stack

CLEAR(A) & HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

Goal stack

Which operator is useful for ON(C,A)?

STACK(*x*,*y*)
P: CLEAR(*y*) & HOLDING(*x*)
D: CLEAR(*y*) & HOLDING(*x*)
A: ARMEMPTY & ON(*x*,*y*)

ON(x,y) and **ON(C,A)** are unified with a unifier: (x=C, y=A)
This is then applied to the full operator rule.

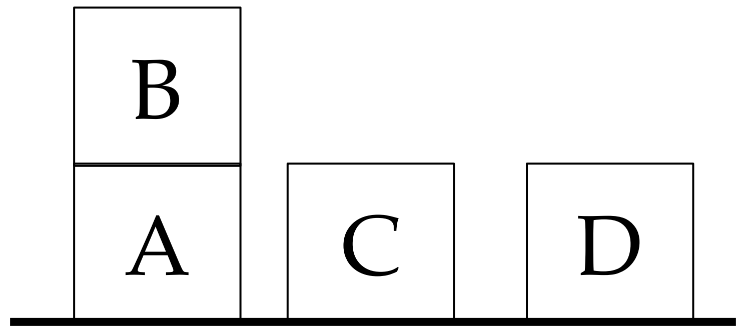
STACK(*C*,*A*)
P: CLEAR(*A*) & HOLDING(*C*)
D: CLEAR(*A*) & HOLDING(*C*)
A: ARMEMPTY & ON(*C*,*A*)

Neither CLEAR(A) or HOLDING(C) are satisfied. What shall we do?

Set up as new subgoals

Planning

Start



Next Step – Setup as new subgoals

CLEAR(A)
HOLDING(C)
CLEAR(A) & HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

Goal stack

Which operator is useful for CLEAR(A)?

STACK(*x*,*y*)
P: CLEAR(*y*) & HOLDING(*x*)
D: CLEAR(*y*) & HOLDING(*x*)
A: ARMEMPTY & ON(*x*,*y*)

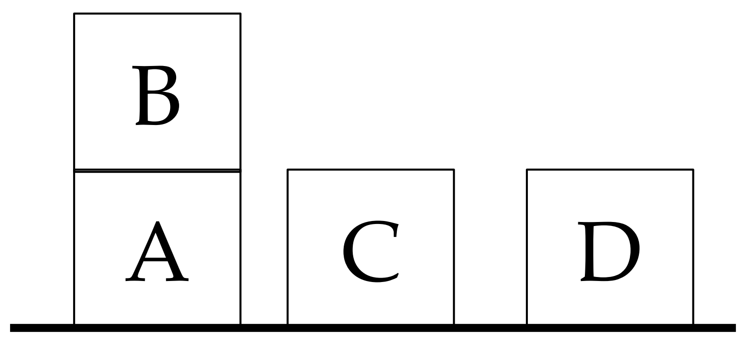
PICKUP(*x*)
P: CLEAR(*x*)&ONTABLE(*x*)& ARMEMPTY
D: ONTABLE(*x*) & ARMEMPTY
A: HOLDING(*x*)

UNSTACK(*x*,*y*)
P: ON(*x*,*y*) & CLEAR(*x*) & ARMEMPTY
D: ON(*x*,*y*) & ARMEMPTY
A: HOLDING(*x*) & CLEAR(*y*)

PUTDOWN(*x*)
P: HOLDING(*x*)
D: HOLDING(*x*)
A: ONTABLE(*x*) & ARMEMPTY

Planning

Start



Next Step – Setup as new subgoals

CLEAR(A)
HOLDING(C)
CLEAR(A) & HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

Goal stack

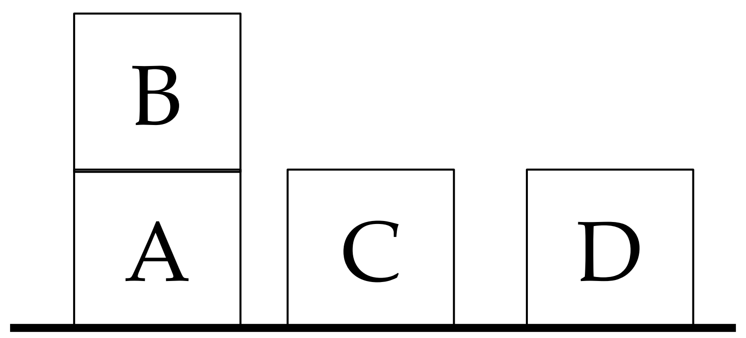
Which operator is useful for CLEAR(A)?

UNSTACK(*x*,*y*)
P: ON(*x*,*y*) & CLEAR(*x*) & ARMEMPTY
D: ON(*x*,*y*) & ARMEMPTY
A: HOLDING(*x*) & CLEAR(*y*)

What value for x and y in UNSTACK(x,y)?

To unify Clear(A), Clear(y)
y=A
What about x?

Start



Planning

Next Step – Setup as new subgoals

CLEAR(A)

HOLDING(C)

CLEAR(A) & HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

Which operator is useful for CLEAR(A)?

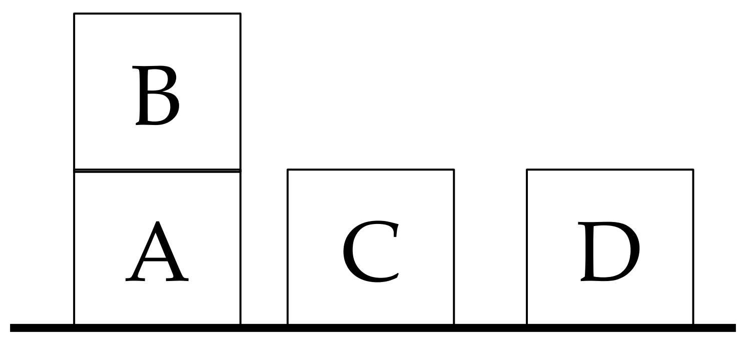
UNSTACK(*x,y*)
P: ON(*x,y*) & CLEAR(*x*) & ARMEMPTY
D: ON(*x,y*) & ARMEMPTY
A: HOLDING(*x*) & CLEAR(*y*)

y=A, x=B

UNSTACK(*B,A*)
P: ON(*B,A*) & CLEAR(*B*) & ARMEMPTY
D: ON(*B,A*) & ARMEMPTY
A: HOLDING(*B*) & CLEAR(*A*)

Planning

Start



Next Step – Setup as new subgoals

~~**ON(B,A) & CLEAR(B) & ARMEMPTY**~~

UNSTACK(B,A)

HOLDING(C)

CLEAR(A) & HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

Which operator is useful for CLEAR(A)?

UNSTACK(*x*,*y*)
P: ON(*x*,*y*) & CLEAR(*x*) & ARMEMPTY
D: ON(*x*,*y*) & ARMEMPTY
A: HOLDING(*x*) & CLEAR(*y*)

y=A, x=B

UNSTACK(*B*,*A*)
P: ON(*B*,*A*) & CLEAR(*B*) & ARMEMPTY
D: ON(*B*,*A*) & ARMEMPTY
A: HOLDING(*B*) & CLEAR(*A*)

Are the pre-conditions satisfied?

Yes. You can remove them now.

Planning

Next Step – carry out the action UNSTACK(B,A)

~~**UNSTACK(B,A)**~~

HOLDING(C)

CLEAR(A) & HOLDING(C)

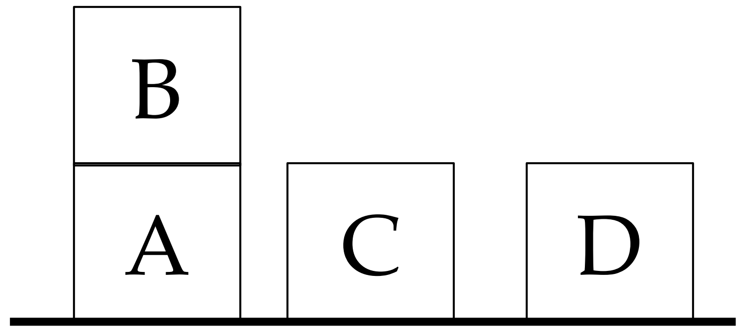
STACK(C,A)

ON(B,D)

ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

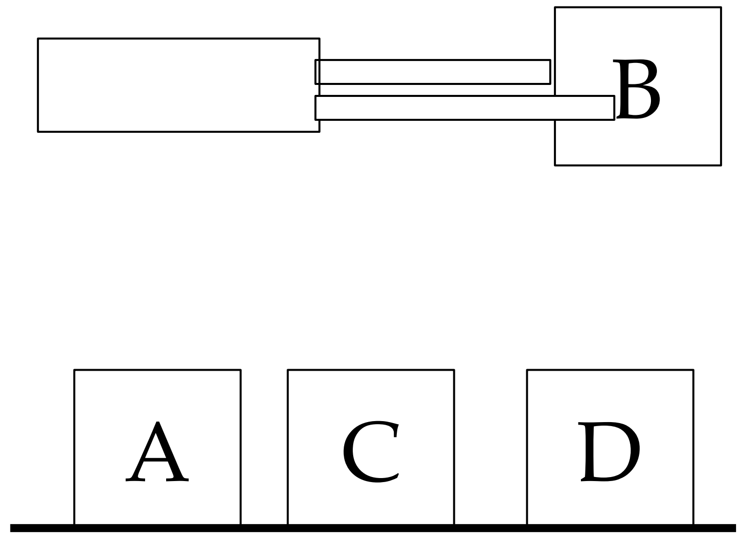
Start S0



State S0:
ON(B,A) &
CLEAR(B) &
CLEAR(C) &
CLEAR(D) &
ONTABLE(A) &
ONTABLE(C) &
ONTABLE(D) &
ARMEMPTY

UNSTACK(*B*,*A*)
P: ON(*B*,*A*) & CLEAR(*B*) & ARMEMPTY
D: ON(*B*,*A*) & ARMEMPTY
A: HOLDING(*B*) & CLEAR(*A*)

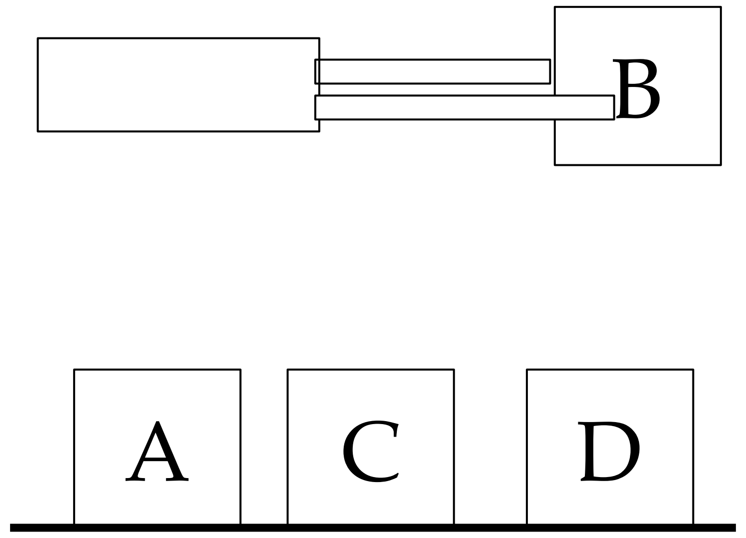
S1



State S1:
CLEAR(A) &
CLEAR(B) &
CLEAR(C) &
CLEAR(D) &
ONTABLE(A) &
ONTABLE(C) &
ONTABLE(D) &
HOLDING(B)

Planning

S1



Next Step – how to satisfy HOLDING(C)?

Two choices: UNSTACK, **PICKUP**

STACK(*x*,*y*)
P: CLEAR(*y*) & HOLDING(*x*)
D: CLEAR(*y*) & HOLDING(*x*)
A: ARMEMPTY & ON(*x*,*y*)

PICKUP(*x*)
P: CLEAR(*x*)&ONTABLE(*x*)& ARMEMPTY
D: ONTABLE(*x*) & ARMEMPTY
A: HOLDING(*x*)

UNSTACK(*x*,*y*)
P: ON(*x*,*y*) & CLEAR(*x*) & ARMEMPTY
D: ON(*x*,*y*) & ARMEMPTY
A: HOLDING(*x*) & CLEAR(*y*)

PUTDOWN(*x*)
P: HOLDING(*x*)
D: HOLDING(*x*)
A: ONTABLE(*x*) & ARMEMPTY

HOLDING(C)

CLEAR(A) & HOLDING(C)

STACK(C,A)

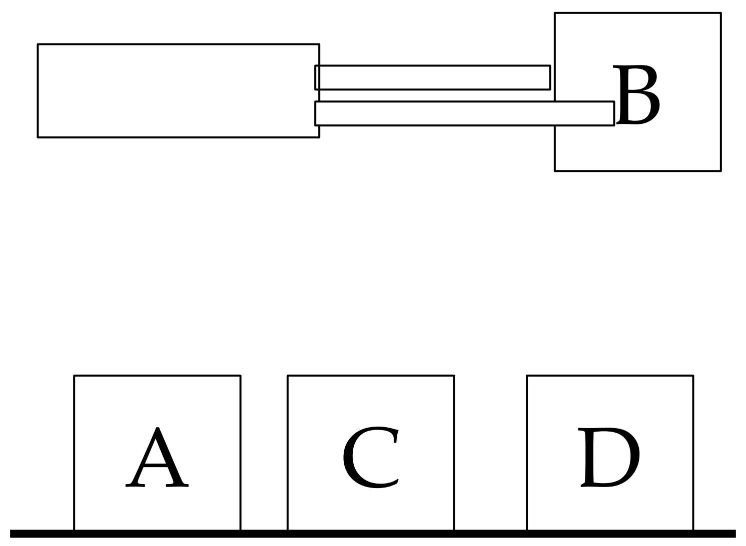
ON(B,D)

ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

Goal stack

Planning

S1



Next Step – how to satisfy HOLDING(C)? PICKUP(C)

CLEAR(C) & ONTABLE(C) & ARMEMPTY
PICKUP(C)
CLEAR(A) & HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

STACK(*x*,*y*)
P: CLEAR(*y*) & HOLDING(*x*)
D: CLEAR(*y*) & HOLDING(*x*)
A: ARMEMPTY & ON(*x*,*y*)

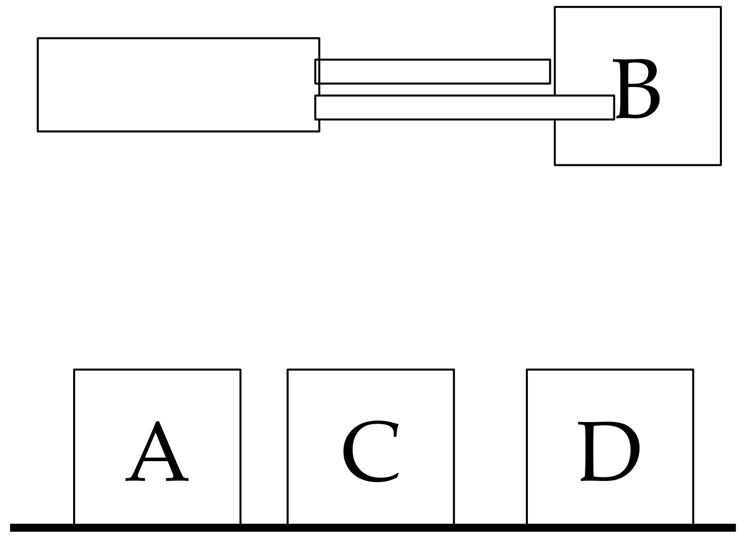
PICKUP(*x*)
P: CLEAR(*x*)&ONTABLE(*x*)& ARMEMPTY
D: ONTABLE(*x*) & ARMEMPTY
A: HOLDING(*x*)

UNSTACK(*x*,*y*)
P: ON(*x*,*y*) & CLEAR(*x*) & ARMEMPTY
D: ON(*x*,*y*) & ARMEMPTY
A: HOLDING(*x*) & CLEAR(*y*)

PUTDOWN(*x*)
P: HOLDING(*x*)
D: HOLDING(*x*)
A: ONTABLE(*x*) & ARMEMPTY

Planning

S1



Next Step - Which operator is useful for ARMEMPTY?

ARMEMPTY

CLEAR(C) & ONTABLE(C) & ARMEMPTY

PICKUP(C)

CLEAR(A) & HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) & ON(B,D) &

ONTABLE(A) & ONTABLE(D)

Goal stack

STACK(*x*,*y*)
P: CLEAR(*y*) & HOLDING(*x*)
D: CLEAR(*y*) & HOLDING(*x*)
A: ARMEMPTY & ON(*x*,*y*)

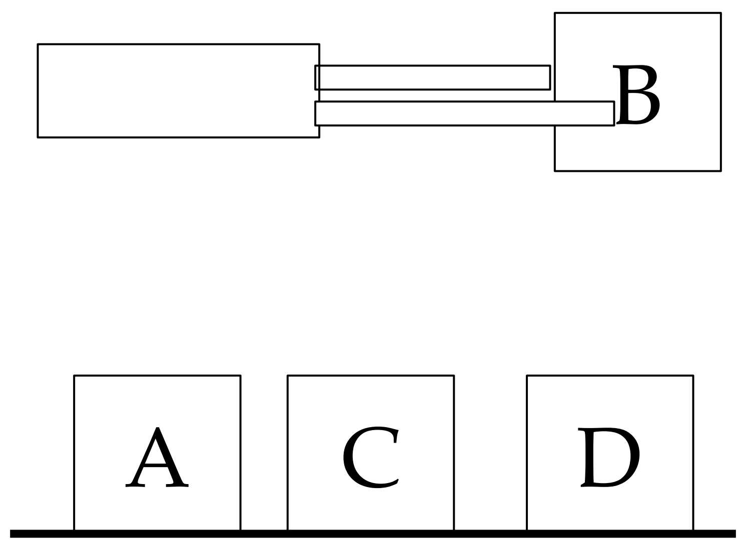
UNSTACK(*x*,*y*)
P: ON(*x*,*y*) & CLEAR(*x*) & ARMEMPTY
D: ON(*x*,*y*) & ARMEMPTY
A: HOLDING(*x*) & CLEAR(*y*)

PICKUP(*x*)
P: CLEAR(*x*)&ONTABLE(*x*)& ARMEMPTY
D: ONTABLE(*x*) & ARMEMPTY
A: HOLDING(*x*)

PUTDOWN(*x*)
P: HOLDING(*x*)
D: HOLDING(*x*)
A: ONTABLE(*x*) & ARMEMPTY

Planning

S1



Next Step - Which operator is useful for ARMEMPTY?

STACK and PUTDOWN

ARMEMPTY

CLEAR(C) & ONTABLE(C) & ARMEMPTY

PICKUP(C)

CLEAR(A) & HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) & ON(B,D) &

ONTABLE(A) & ONTABLE(D)

Goal stack

STACK(x,y)
- P: CLEAR(y) & HOLDING(x)
- D: CLEAR(y) & HOLDING(x)
- A: ARMEMPTY & ON(x,y)

PICKUP(x)
- P: CLEAR(x)&ONTABLE(x)& ARMEMPTY
- D: ONTABLE(x) & ARMEMPTY
- A: HOLDING(x)

UNSTACK(x,y)
- P: ON(x,y) & CLEAR(x) & ARMEMPTY
- D: ON(x,y) & ARMEMPTY
- A: HOLDING(x) & CLEAR(y)

PUTDOWN(x)
- P: HOLDING(x)
- D: HOLDING(x)
- A: ONTABLE(x) & ARMEMPTY

STACK(B,*y*)

P: CLEAR(*y*) & HOLDING(B)
D: CLEAR(*y*) & HOLDING(B)
A: ARMEMPTY & ON(B,*y*)

PUTDOWN(B)

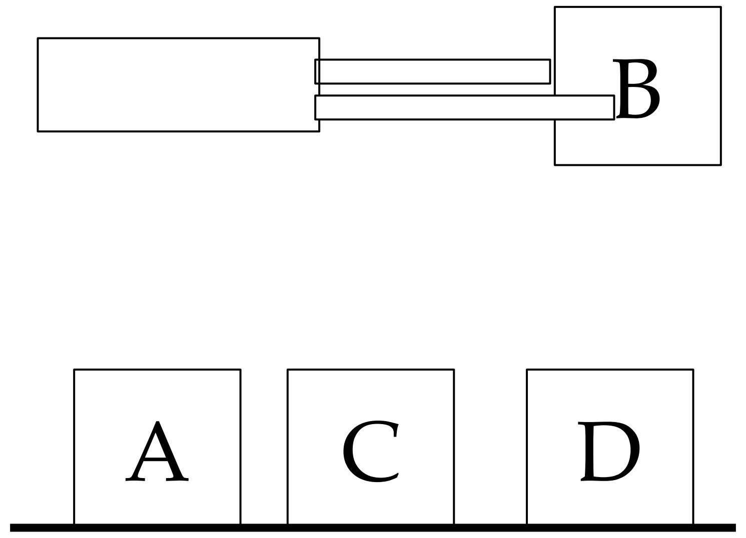
P: HOLDING(B)
D: HOLDING(B)
A: ONTABLE(B) & ARMEMPTY

Another heuristic: choose one which can satisfy multiple goals – Killing two birds with one stone.

Which one to choose?

Planning

S1



Next Step - select STACK(B,D)

CLEAR(D) & HOLDING(B)
STACK(B,D)
CLEAR(C) & ONTABLE(C) & ARMEMPTY
PICKUP(C)
CLEAR(A) & HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

STACK(*x*,*y*)
P: CLEAR(*y*) & HOLDING(*x*)
D: CLEAR(*y*) & HOLDING(*x*)
A: ARMEMPTY & ON(*x*,*y*)

PICKUP(*x*)
P: CLEAR(*x*)&ONTABLE(*x*)&
ARMEMPTY
D: ONTABLE(*x*) & ARMEMPTY
A: HOLDING(*x*)

UNSTACK(*x*,*y*)
P: ON(*x*,*y*) & CLEAR(*x*) &
ARMEMPTY
D: ON(*x*,*y*) & ARMEMPTY
A: HOLDING(*x*) & CLEAR(*y*)

PUTDOWN(*x*)
P: HOLDING(*x*)
D: HOLDING(*x*)
A: ONTABLE(*x*) & ARMEMPTY

STACK(B,*D*)

P: CLEAR(*D*) & HOLDING(B)
D: CLEAR(*D*) & HOLDING(B)
A: ARMEMPTY & ON(B,*D*)

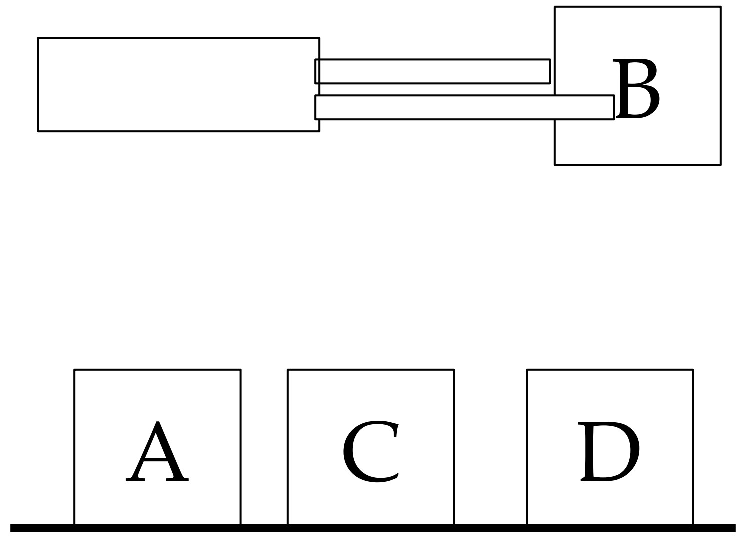
Planning

Next Step - Executing STACK(B,D)

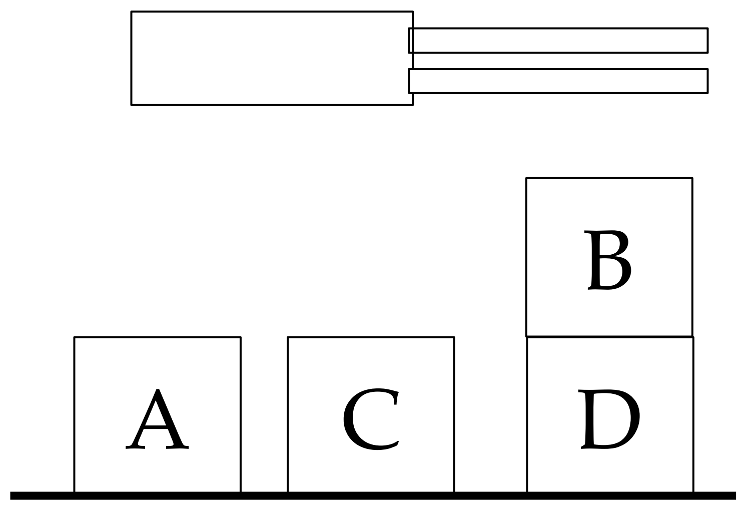
~~CLEAR(D) & HOLDING(B)~~
~~**STACK(B,D)**~~
CLEAR(C) & ONTABLE(C) & ARMEMPTY
PICKUP(C)
CLEAR(A) & HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

S1

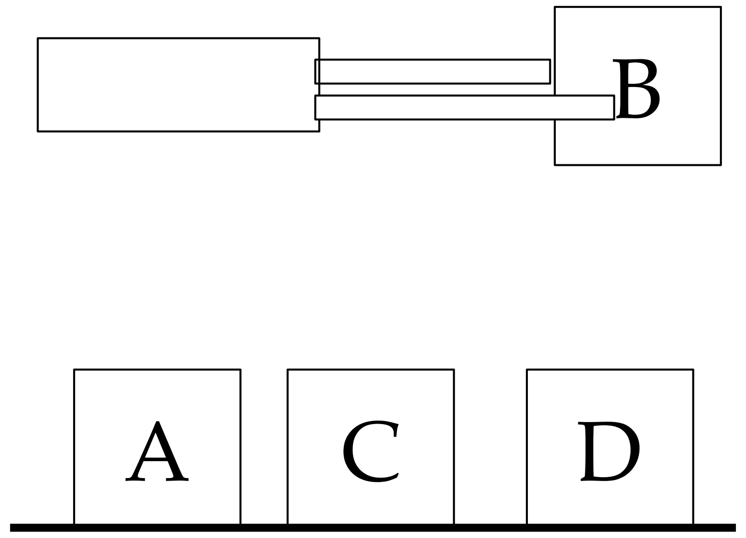


S2



Planning

S1

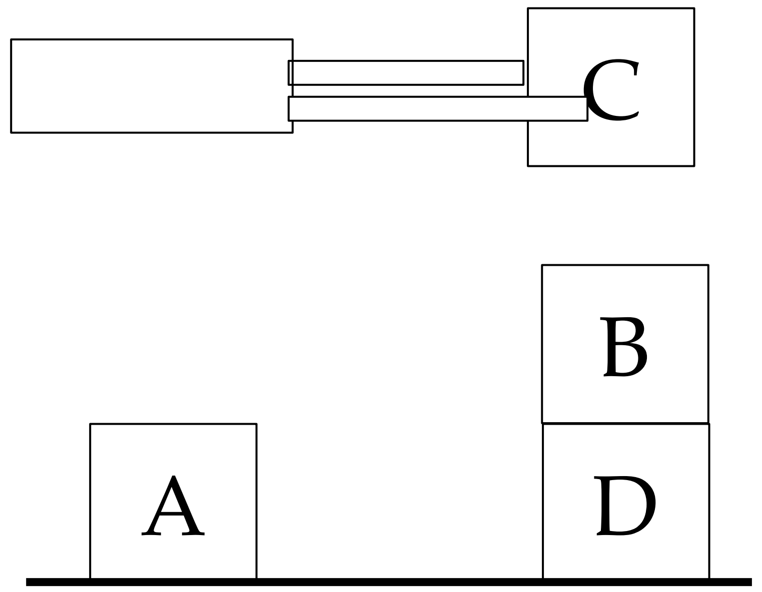


Next Step - Executing PICKUP(C)

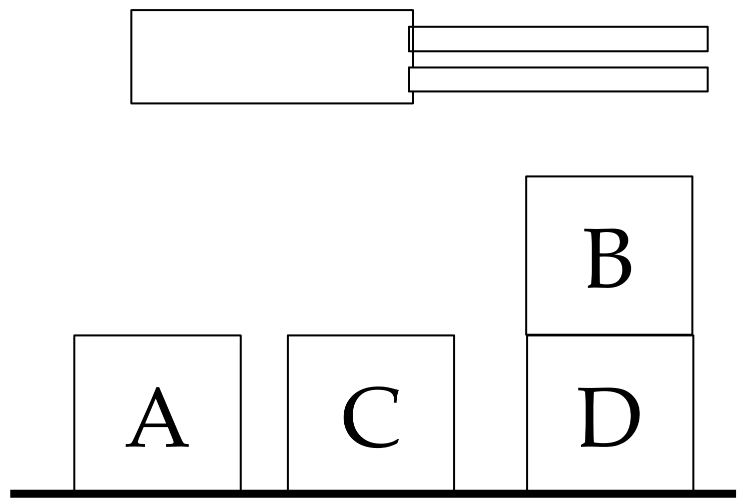
~~CLEAR(C) & ONTABLE(C) & ARMEMPTY~~
~~**PICKUP(C)**~~
CLEAR(A) & HOLDING(C)
STACK(C,A)
ON(B,D)
ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

S3

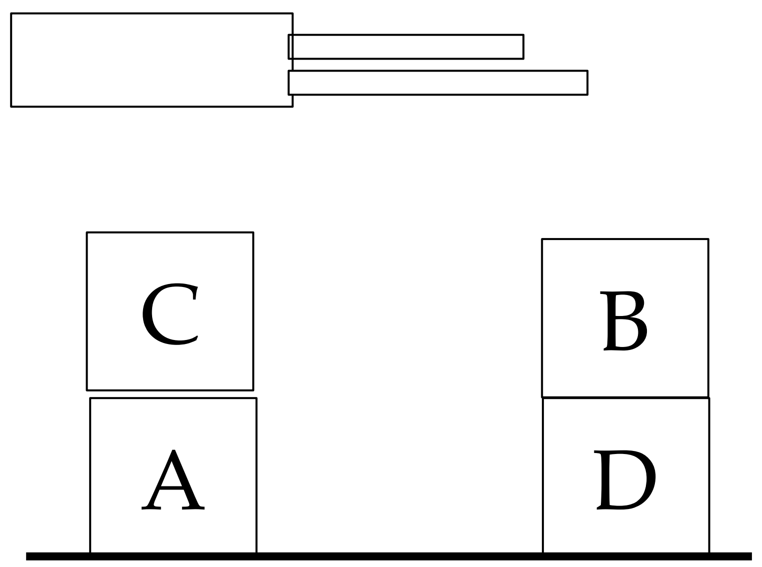


S2

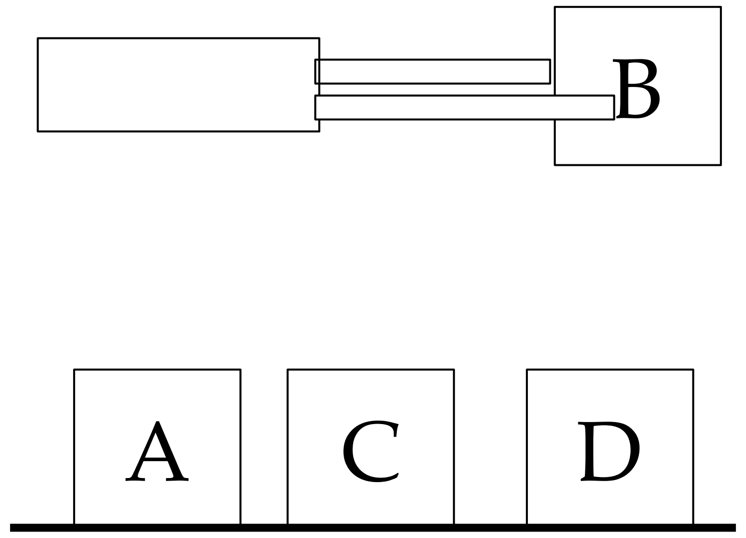


Planning

S4

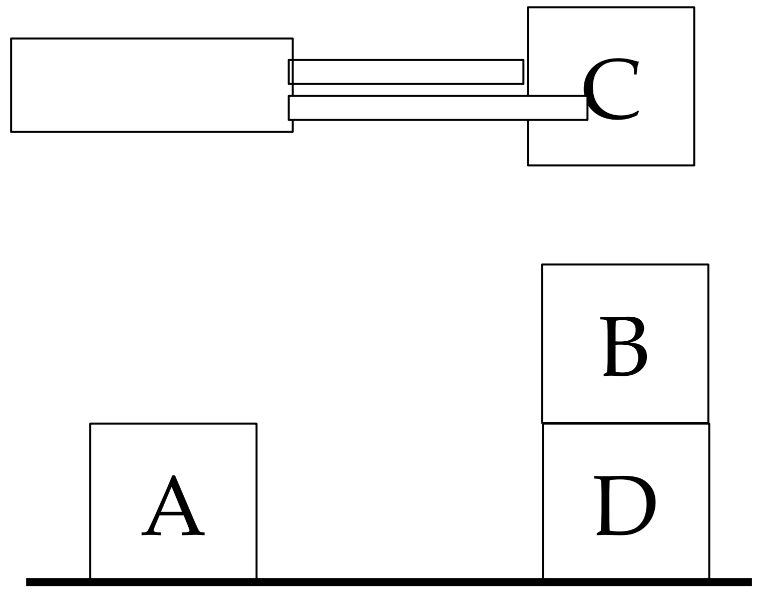


S1

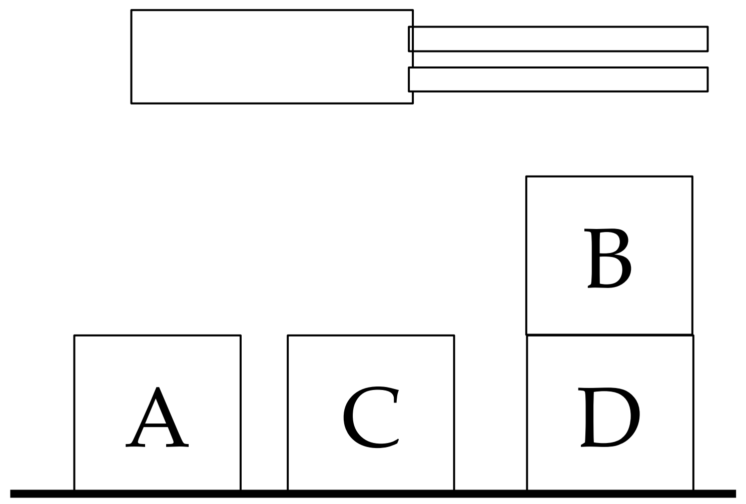


Next Step - Executing STACK(C,A)

S3



S2



~~CLEAR(A) & HOLDING(C)~~
~~STACK(C,A)~~
ON(B,D)
ON(C,A) & ON(B,D) &
ONTABLE(A) & ONTABLE(D)

Goal stack

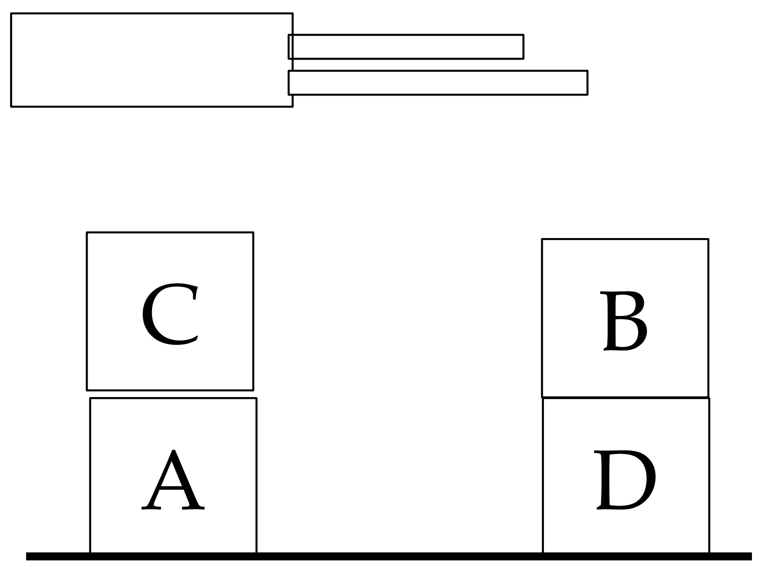
Planning

Next Step - Mission Accomplished

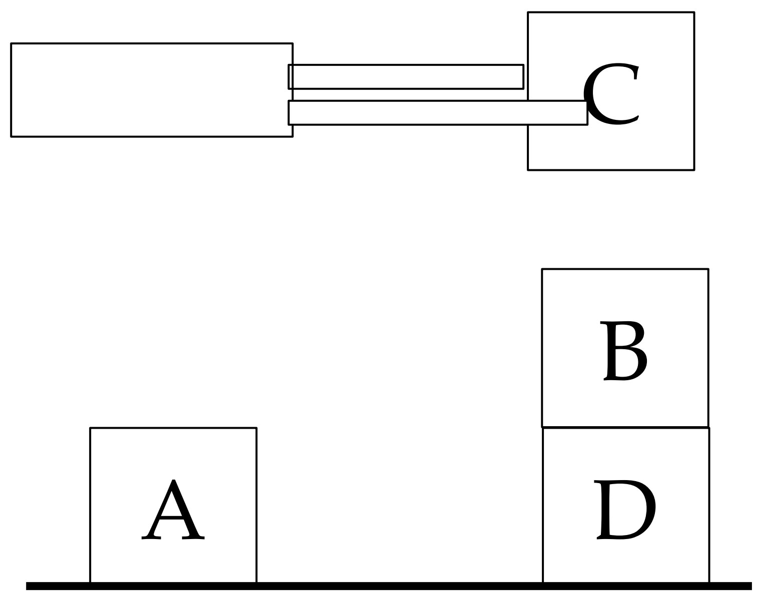
~~ON(B,D)~~
~~ON(C,A) & ON(B,D) &~~
~~ONTABLE(A) & ONTABLE(D)~~

Goal stack

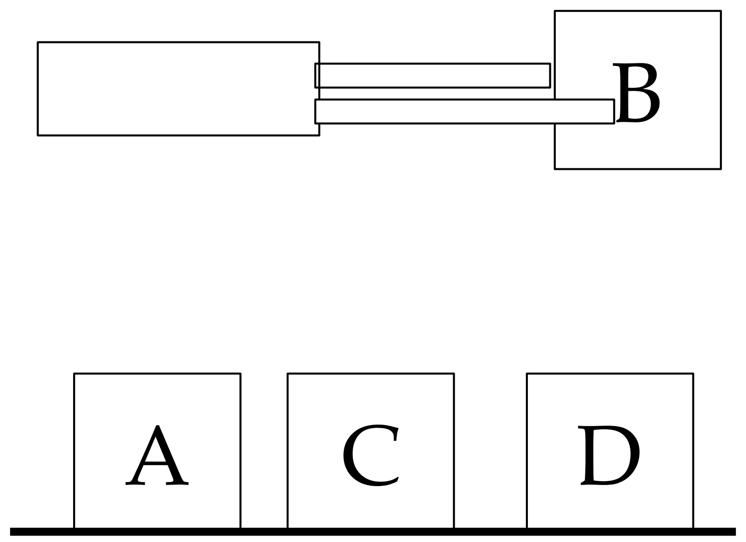
S4



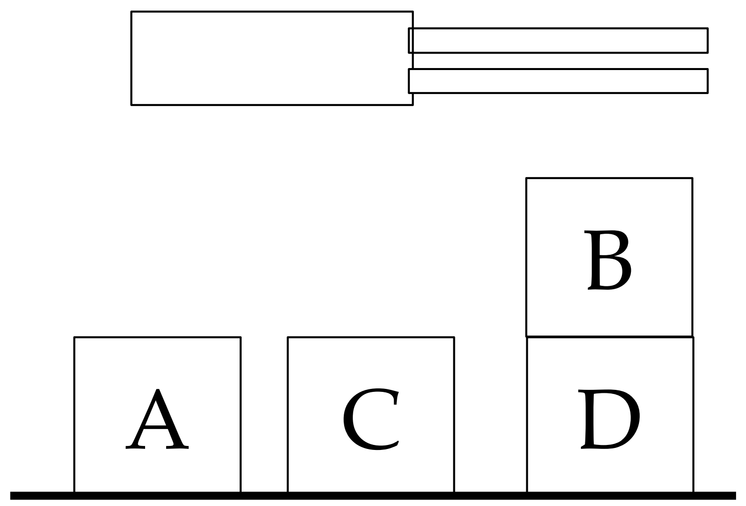
S3



S1



S2



Planning

Next Step - Mission Accomplished

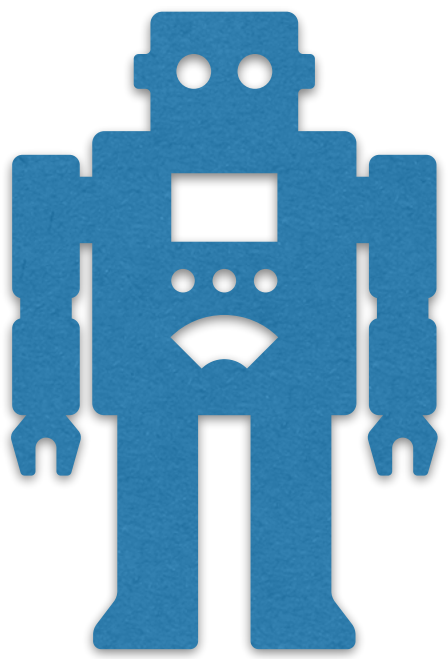
The plan is

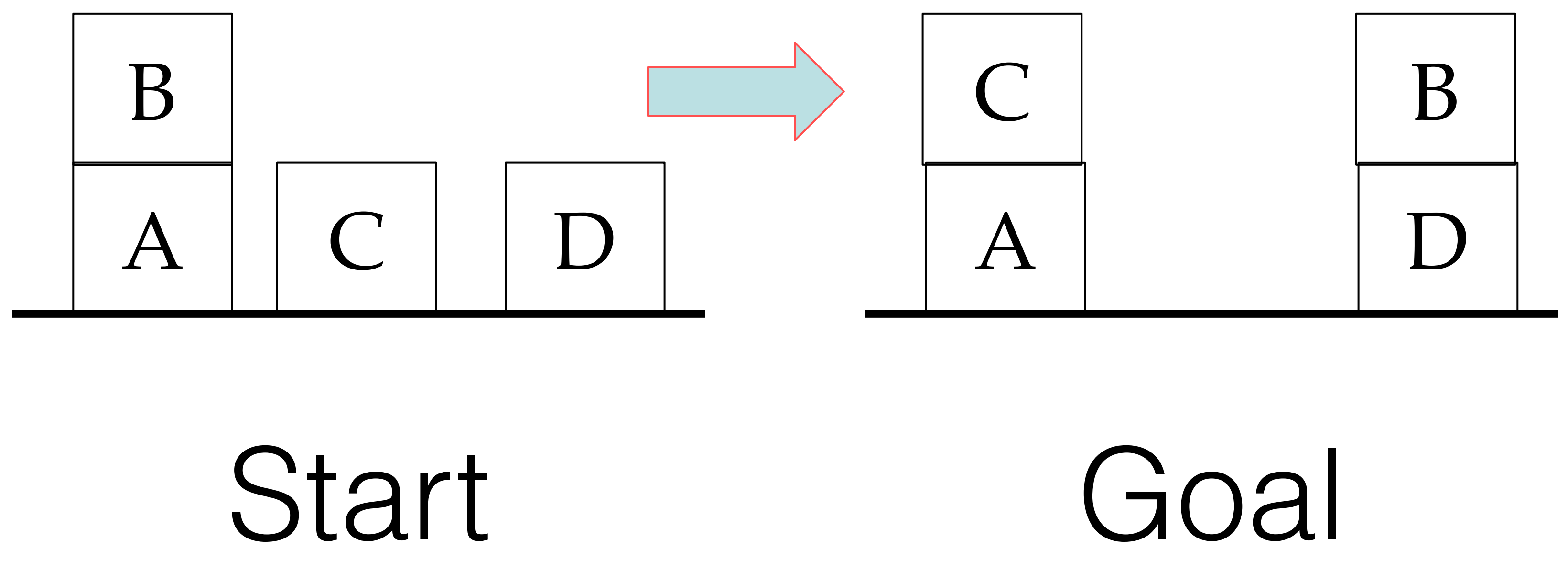
UNSTACK(B,A)

STACK(B,D)

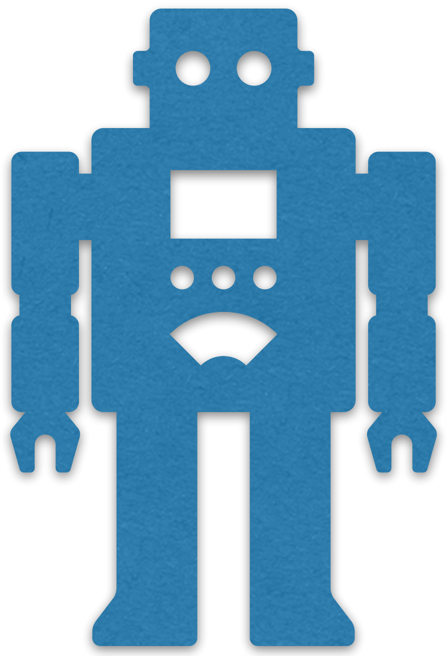
PICKUP(C)

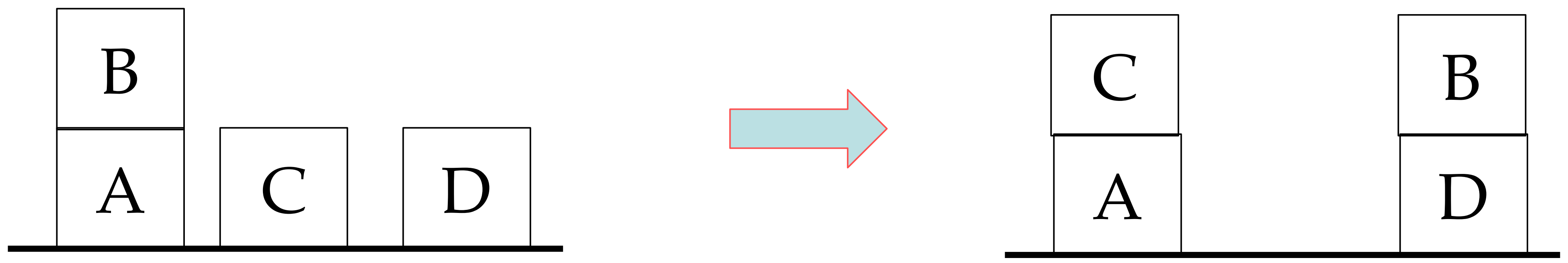
STACK(C,A)





Planning





Start

ON(B,A) & CLEAR(B) & CLEAR(C) & CLEAR(D) & ONTABLE(A) & ONTABLE(C) & ONTABLE(D) & ARMEMPTY

Goal

ON(C,A) & ON(B,D) & ONTABLE(A) & ONTABLE(D)

To form a plan

1. Input a list of goals, G, to be achieved
2. Unless input list is empty, pick one goal
3. Find a suitable action or a set of actions that will enable us to accomplish that goal
4. Execute actions, if successful repeat step 2.
5. Repeat Step 3.

Planning applications

- A hundred million miles from Earth, NASA's Remote Agent program became the first on-board autonomous planning program to control the scheduling of operations for a spacecraft (Jonsson et al., 2000). Remote Agent generated plans from high-level goals specified from the ground and monitored the execution of those plans—detecting, diagnosing, and recovering from problems as they occurred. Today, the EUROPA planning toolkit (Barreiro et al., 2012) is used for daily operations of NASA's Mars rovers and the SEXTANT system (Winternitz, 2017) allows autonomous navigation in deep space, beyond the global GPS system.

- During the Persian Gulf crisis of 1991, U.S. forces deployed a Dynamic Analysis and Replanning Tool, DART (Cross and Walker, 1994), to do automated logistics planning and scheduling for transportation. This involved up to 50,000 vehicles, cargo, and people at a time, and had to account for starting points, destinations, routes, transport capacities, port and airfield capacities, and conflict resolution among all parameters. The Defense Advanced Research Project Agency (DARPA) stated that this single application more than paid back DARPA's 30-year investment in AI.

Summary

- Means-ends analysis allows us to do intelligent planning. One way to implement the idea is to use goal stack planning.
- STRIPS-style Representation. Actions are described, using a simple P-D-A list.
- Within each action, one has to decide the order in which to carry out the steps needed.
- Sometimes there are more than one actions that one can do to achieve a given goal. Use of heuristics will be useful.
- Select the "wrong" action does not necessary mean failure (unlike a general search problem); you may just end up with a weird plan or a longer way to do things!

References

- **Artificial Intelligence: A Modern Approach**
  http://aima.cs.berkeley.edu

- STRIPS online tutorial:
  https://github.com/primaryobjects/strips/tree/master

- PROLOG
  https://www.swi-prolog.org

- **General Game Playing**
  http://logic.stanford.edu/ggp/homepage/notes.php
  Best GGP Textbook by Michael Genesereth and Michael Thielscher