# Planning

# Automated Planning

## Automated Planning

- Given:

- Find:

# Automated Planning

## Automated Planning

- Given:
  - an initial state
  - a set of actions you can perform
  - a (set of) state(s) to achieve (goal)
- Find:
  - a plan: a partially- or totally-ordered set of actions needed to achieve the goal from the initial state

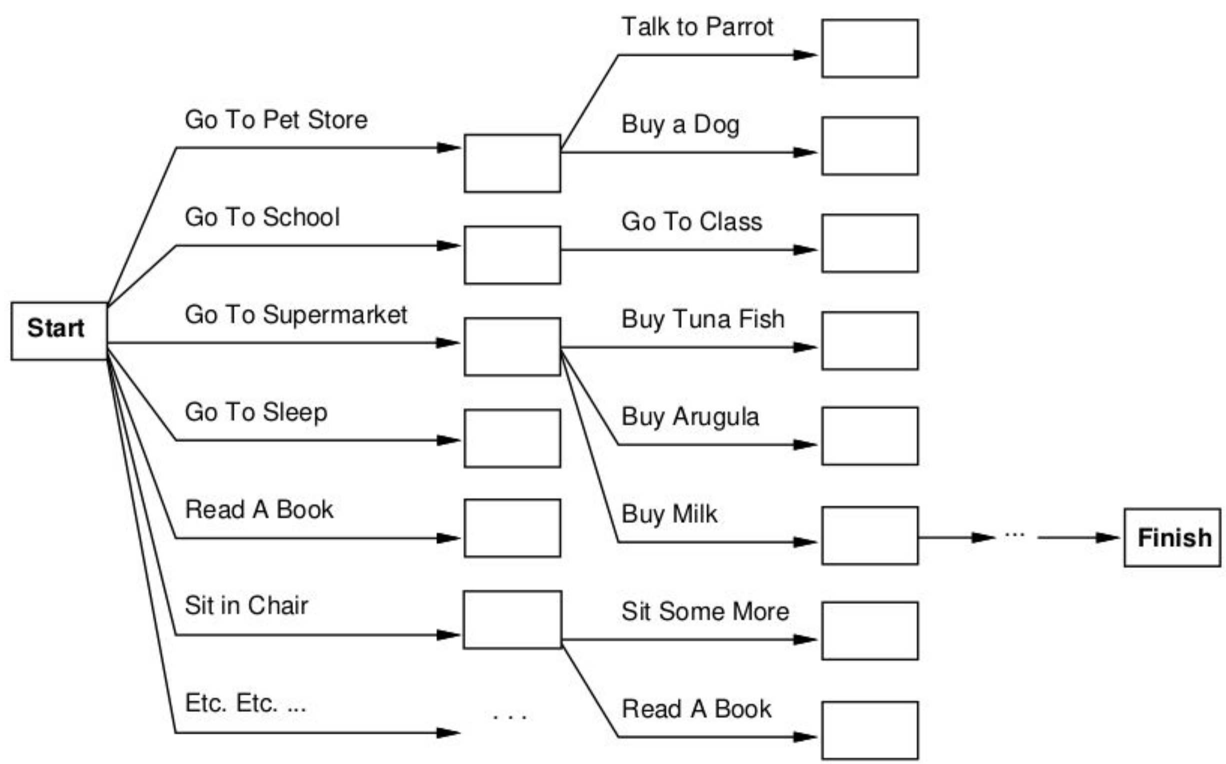
# Introduction to Planning

- Planning is required for every task (Job).
  - Example: reaching a particular destination requires planning. First we should find the best route and then identify a set of actions to be done at a particular time
- Planning aims to determine a **set of actions** to be performed, in an agent environment based on the perception to achieve the goal.
- A plan is simply a sequence of actions.

# Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*

Standard search algorithms seem to fail miserably:




After-the-fact heuristic/goal test inadequate

# Introduction to Planning

- Two types of planning
  - **Classical planning** where we have fully observable, deterministic, finite, static and discrete environments.
  - **Nonclassical planning** where we have partially observable or stochastic environments.

# Introduction to Planning

- For any planning system, following informations are needed.
- Domain description.
- **Action** specification:
  - Each action has its own set of **preconditions**
  - All the preconditions to be satisfied before performing the action.
  - An action results in a **effects**, which can be either positive or negative.
- **Goal** description.

# Representation of States

- Planners decompose the agent world into logical conditions, and represent a state as a conjunction of positive literals.
- In the state descriptions, Literals must be ground and function free, e.g., At (x, y) or At (Father(Red), Sydney) are not allowed.
- The closed world assumption, that is any conditions that are not mentioned in a state are assumed false

# Representation of Goals.

- A goal is represented as a conjunction of positive ground literals. E.g., Rich ^ Satisfied.
- A propositional state s, satisfies a goal G, if s contains all the atoms (i.e., ground literals) in G.
- For example,
    - **G** Goal state = Rich ^ Satisfied
    - **s** propositional state = Rich ^ Satisfied ^ Happy
    - **s** satisfies the goal state.

# Action Schema

- Actions are represented with *preconditions* and *effects*.
- Action Schema represents different actions. It consists of three parts:
  - **Action name** and **parameter** list, for example, Fly(p, from, to), Fly is action name, and parameters are from and to.
  - **Precondition**: a conjunction of function-free positive literals, and it must be true, before the action can be executed.
  - **Effect**: a conjunction of function-free literals, describing how the state changes, when the action is executed.

# Action Schema

- For example, let us describe an action for flying a plane

Action : Fly(p, from, to)

PRECOND: At (p, from) ^ Plane(p) ^ Airport(from) ^ Airport(to)

EFFECT : At(p, from) ^ At (p, to)

**Action Schema**

# Action Schema

Effect can be divided as

- Add list for positive literals and
- Delete list for negative literals

# Applicable Action

- An action is applicable in any state, iff that satisfies the precondition, otherwise, the action has no effect.
- For example

  Action : Fly(p, from, to)

  Precondition: At (p, from) ^ Plane(p) ^ Airport(from) ^ Airport(to)
- Consider State:

  **At** (P1, JP) ^ At (P2, DL) ^ **Plane**(P1) ^ Plane(P2) ^ **Airport**(JP) ^ **Airport**(DL).
- Above state satisfy Precondition with {p=P1, from=JP, to=DL}
- Thus, Fly(P1, JP, DL) is applicable action.

# PDDL

# Planning Domain Definition Language

- The Planning Domain Definition Language (PDDL) is an attempt to standardize Artificial Intelligence (AI) planning languages.
- It was first developed by Drew McDermott and his colleagues in 1998 (inspired by Stanford Research Institute Problem Solver, known by its acronym STRIPS)
- **PDDL** allows us to express actions with one action schema.

# A language for planning: PDDL

- A state is a conjunction of fluents: ground, function-less atoms
  - ex: $Poor \wedge Unknown$, $At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$
  - ex of non-fluents: $At(x, y)$ (non ground), $\neg Poor$ (negated), $At(Father(Fred), Sydney)$ (not function-less)
  - closed-world assumption: all non-mentioned fluents are false
  - unique names assumption: distinct names refer to distinct objects

# A language for planning: PDDL

- A state is a conjunction of fluents: ground, function-less atoms
  - ex: $Poor \wedge Unknown$, $At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$
  - ex of non-fluents: $At(x, y)$ (non ground), $\neg Poor$ (negated), $At(Father(Fred), Sydney)$ (not function-less)
  - closed-world assumption: all non-mentioned fluents are false
  - unique names assumption: distinct names refer to distinct objects
- Actions are described by a set of action schemata
  - concise description: describe which fluent change

$\Longrightarrow$ the other fluents implicitly maintain their values

# PDDL Example

- Action schema:
  $Action(Fly(p, from, to),$
  $PRECOND:$
  $EFFECT\ :$
- Action instantiation:
  $Action(Fly(P_1, SFO, JFK),$
  $PRECOND:$
  $EFFECT\ :$

# PDDL Example

- Action schema:
  $$\begin{aligned}&Action(Fly(p, from, to),\\ &\quad PRECOND : At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)\\ &\quad EFFECT\ \ : \neg At(p, from) \land At(p, to))\end{aligned}$$
- Action instantiation:
  $$\begin{aligned}&Action(Fly(P_1, SFO, JFK),\\ &\quad PRECOND : At(P_1, SFO) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK)\\ &\quad EFFECT\ \ : \neg At(P_1, SFO) \land At(P_1, JFK))\end{aligned}$$

# PDDL

- Precondition: must hold to ensure the action can be executed
  - defines the states in which the action can be executed
  - action is applicable in state s if the preconditions are satisfied by s

# PDDL

- Precondition: must hold to ensure the action can be executed
    - defines the states in which the action can be executed
    - action is applicable in state s if the preconditions are satisfied by s
- Effect: represent the effects of the action on the world
    - defines the result of executing the action
- Add list (ADD(a)): the positive literals in the action's effects
    - ex: $\{At(p, to)\}$

# PDDL

- Precondition: must hold to ensure the action can be executed
  - defines the states in which the action can be executed
  - action is applicable in state s if the preconditions are satisfied by s
- Effect: represent the effects of the action on the world
  - defines the result of executing the action
- Add list (ADD(a)): the positive literals in the action's effects
  - ex: $\{At(p, to)\}$
- Delete list (DEL(a)): (the fluents in) the negative literals in the action's effects
  - ex: $\{At(p, from)\}$

# PDDL

- Precondition: must hold to ensure the action can be executed
  - defines the states in which the action can be executed
  - action is applicable in state s if the preconditions are satisfied by s
- Effect: represent the effects of the action on the world
  - defines the result of executing the action
- Add list (ADD(a)): the positive literals in the action's effects
  - ex: $\{At(p, to)\}$
- Delete list (DEL(a)): (the fluents in) the negative literals in the action's effects
  - ex: $\{At(p, from)\}$
- Result of action a in state s: RESULT(s,a)$\stackrel{def}{=}$(s$\setminus$DEL(a) $\cup$ ADD(a))
  - start from s
  - remove the fluents that appear as negative literals in effect
  - add the fluents that appear as positive literals in effect
  - ex: $Fly(P_1, SFO, JFK) \Longrightarrow$ remove $At(P_1, SFO)$, add $At(P_1, JFK)$

# PDDL Example

- Action schema:
  $Action(Fly(p, from, to),$
  $\quad PRECOND : At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
  $\quad EFFECT \quad : \neg At(p, from) \wedge At(p, to))$
- Action instantiation:
  $Action(Fly(P_1, SFO, JFK),$
  $\quad PRECOND : At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$
  $\quad EFFECT \quad : \neg At(P_1, SFO) \wedge At(P_1, JFK))$
- $s : At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK) \wedge \ldots$

# PDDL Example

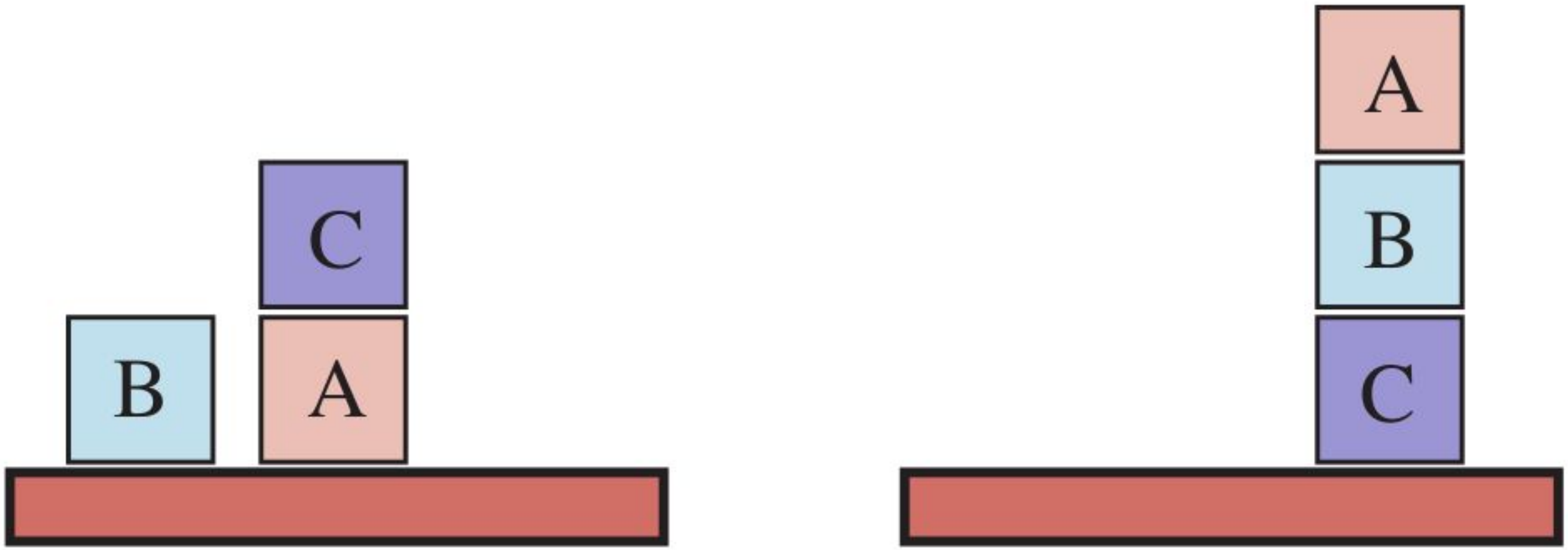
- Action schema:
  $Action(Fly(p, from, to),$
  $\quad PRECOND : At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
  $\quad EFFECT \quad : \neg At(p, from) \wedge At(p, to))$
- Action instantiation:
  $Action(Fly(P_1, SFO, JFK),$
  $\quad PRECOND : At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$
  $\quad EFFECT \quad : \neg At(P_1, SFO) \wedge At(P_1, JFK))$
- $s : At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK) \wedge ...$

$\Longrightarrow$ $s' : At(P_1, JFK) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK) \wedge ...$

Sometimes we want to propositionalize a PDDL problem: replace each action schema with a set of ground actions.

- Ex: $...At\_P_1\_SFO \wedge Plane\_P_1 \wedge Airport\_SFO \wedge Airport\_JFK)...$

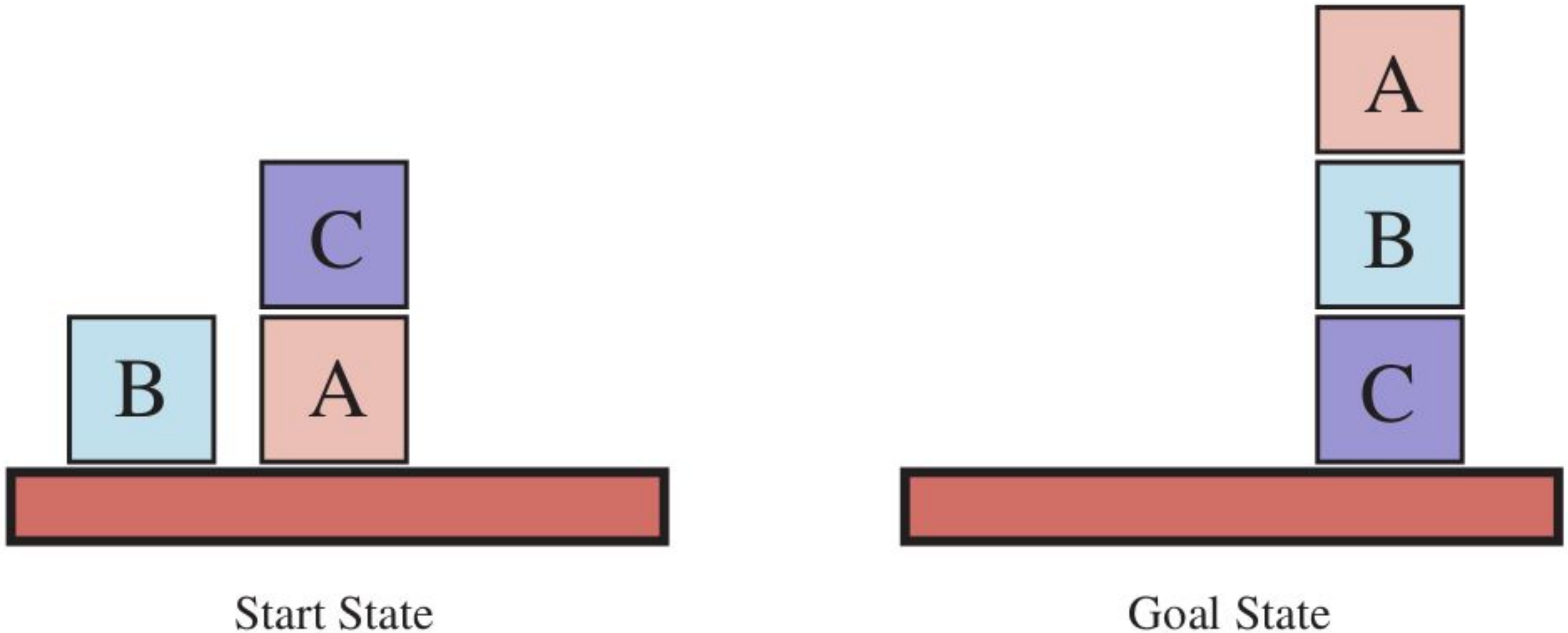
# Blocks-world problem




Start State

Goal State

# Blocks-world problem




Move to table C
Move B on top of C
Move A on Top to B

# Blocks-world problem

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C))$
$Goal(\qquad\qquad\qquad\qquad)$
$Action(Move(b, x, y),$
PRECOND

EFFECT:
$Action(MoveToTable(b, x),$
PRECOND:
EFFECT:

# Blocks-world problem

$$Init(On(A, Table) \land On(B, Table) \land On(C, A)$$
$$\land\ Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C))$$
$$Goal(On(A, B) \land On(B, C))$$
$$Action(Move(b, x, y),$$

PRECOND

EFFECT:

$$Action(MoveToTable(b, x),$$

PRECOND:

EFFECT:

# Blocks-world problem

$$Init(On(A, Table) \land On(B, Table) \land On(C, A)$$
$$\land\ Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C))$$
$$Goal(On(A,B) \land On(B,C))$$
$$Action(Move(b,x,y),$$

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:

$$Action(MoveToTable(b,x),$$

PRECOND:

EFFECT:

# Blocks-world problem

$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, A)$
$\quad \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C))$
$Goal(On(A, B) \wedge On(B, C))$
$Action(Move(b, x, y),$
PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge$
$(b \neq x) \wedge (b \neq y) \wedge (x \neq y),$
EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$
$Action(MoveToTable(b, x),$
PRECOND:
EFFECT:

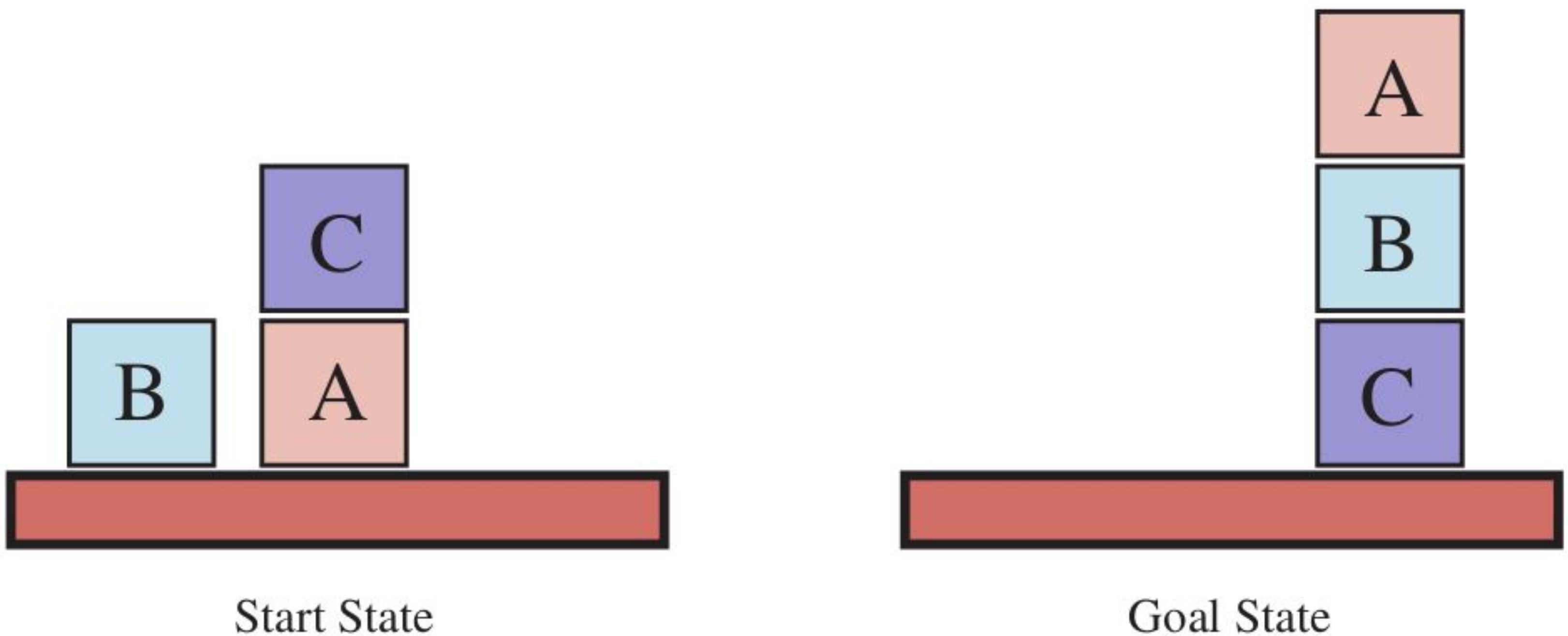
# Blocks-world problem

$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, A)$
$\quad \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C))$
$Goal(On(A, B) \wedge On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge$
$\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y),$
$\quad$ EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x),$
$\quad$ EFFECT:

# Blocks-world problem

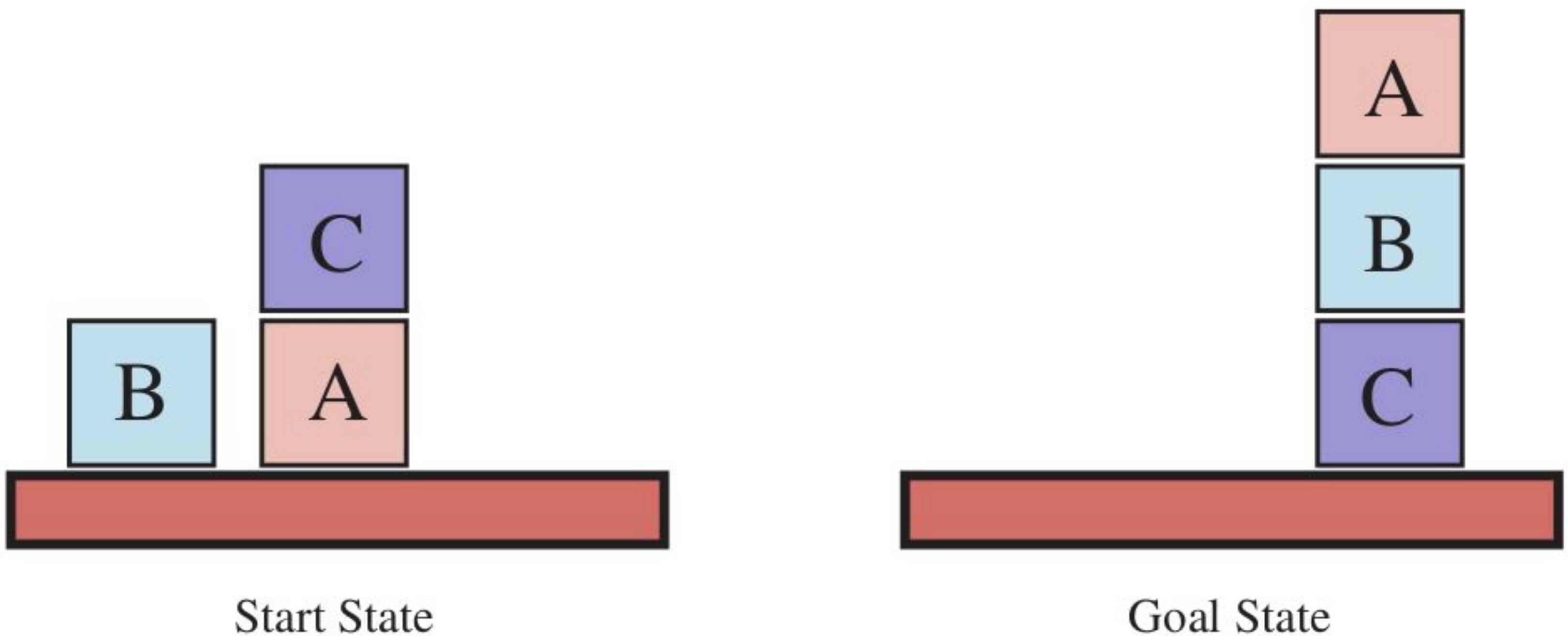
$$Init(On(A, Table) \land On(B, Table) \land On(C, A)$$
$$\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C))$$
$$Goal(On(A, B) \land On(B, C))$$
$$Action(Move(b, x, y),$$
$$\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$$
$$\qquad (b \neq x) \land (b \neq y) \land (x \neq y),$$
$$\quad \text{EFFECT: } On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$$
$$Action(MoveToTable(b, x),$$
$$\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Block(b) \land (b \neq x),$$
$$\quad \text{EFFECT: } On(b, Table) \land Clear(x) \land \neg On(b, x))$$

# Blocks-world problem




solution is the sequence $[MoveToTable(C, A),$ $\qquad\qquad\qquad\qquad\qquad\qquad]$.

# Blocks-world problem

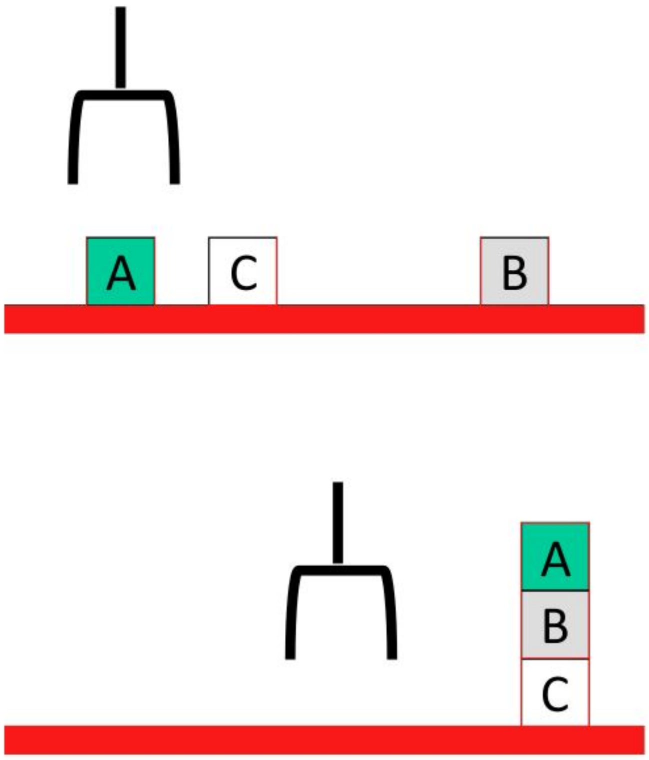



solution is the sequence $[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$.

# STRIPS

# STRIPS

Stanford Research Institute Problem Solver (STRIPS) is an automated planner similar to Planning Domain Definition Language (PDDL).




An example STRIPS problem

- on(X, Y) block X is on block Y
- ontable(X) block X is on the table
- clear(X) no block is on block X
- holding(X) the robot arm is holding X
- armempty the robot arm is not holding anything

# STRIPS

- P: Precondition List - The assertions needed to be true for the operator to be applicable.
- A: Add List - The assertions that became true as ^ consequence of the operator being applied.
- D: Delete List - The assertions that are no longer true as ^ consequence of the operator being applied.

# Actions

- STRIPS representation are action-centric representation.

**PICKUP** (X)

P: ontable (X) ^ clear (X) ^ armempty

A: holding (X)

D: ontable (X) ^ armempty

**PUTDOWN** (X)

P: holding (X)

A: ontable (X) ^ armempty

D: holding (X)

# STRIPS

**UNSTACK** (X, Y)

P: on (X, Y) ^ clear (X) ^ armempty

A: holding (X) ^ clear (Y)

D: on (X, Y) ^ armempty

**STACK** (X, Y)

P: holding (X) ^ clear (Y)

A: on (X, Y) ^ clear (X) ^ armempty

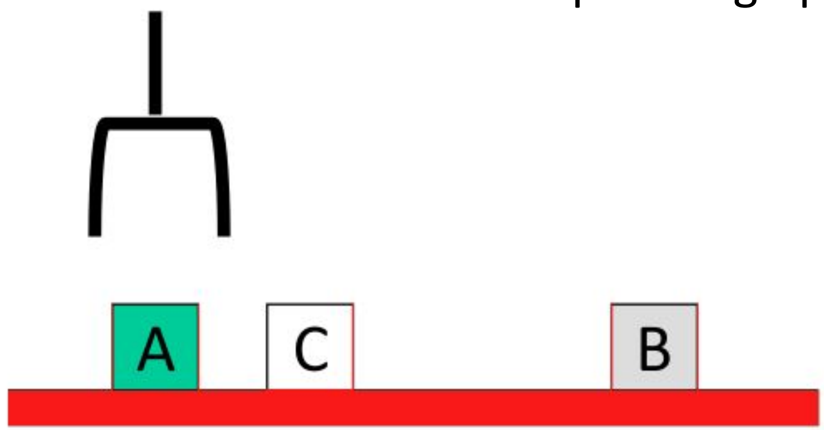
D: holding (X) ^ clear (Y)

# STRIPS

Actions in a STRIPS planning is a ground instance of a planning operator.

Initial state:

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




A plan:

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

Goal:

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