Artificial Intelligence Foundations and Applications

Planning – Part 1

Sudeshna Sarkar
Oct 31 2022

Centre of Excellence in Artificial Intelligence, IIT Kharagpur

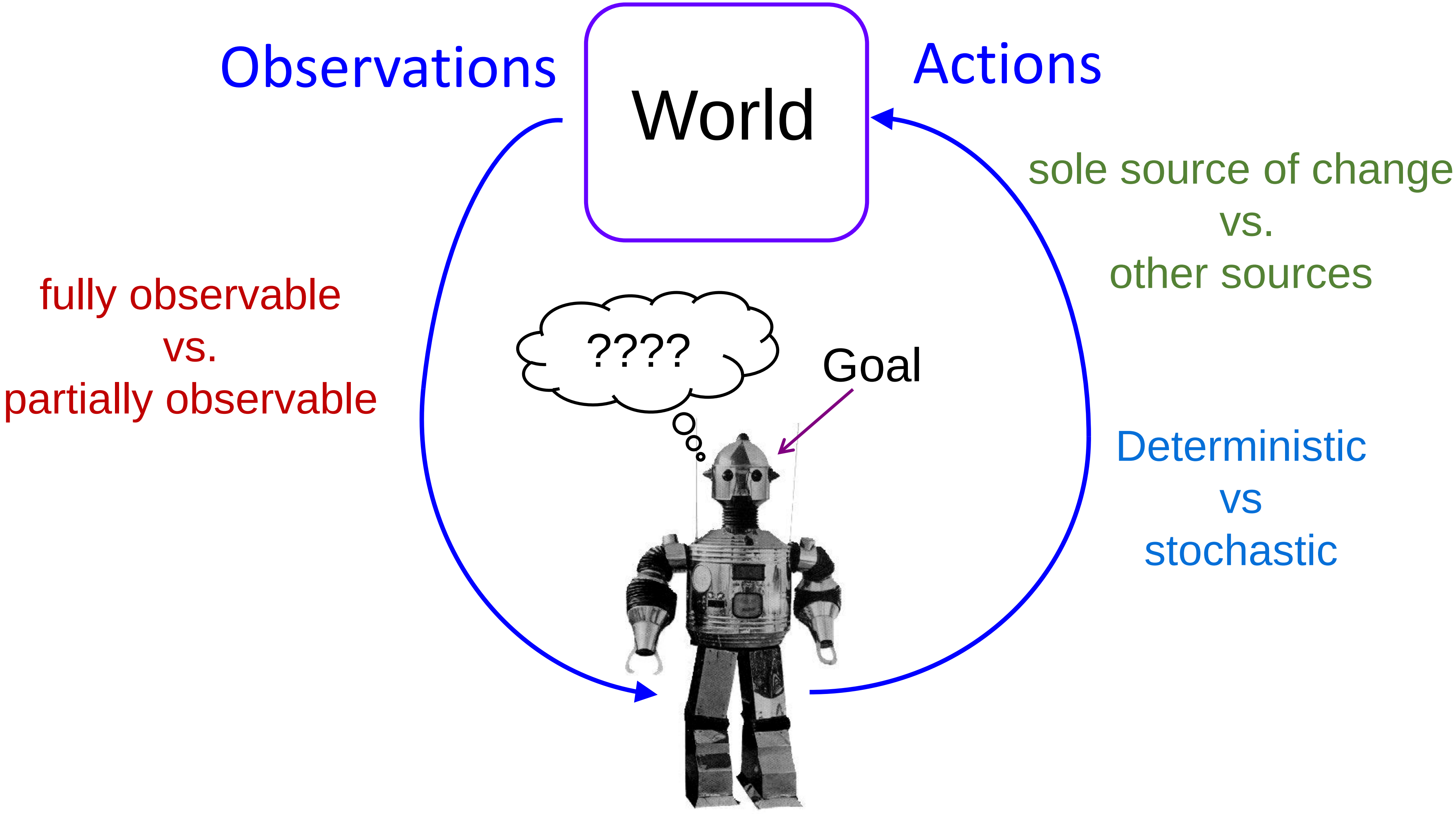
Planning

How can a robot figure out a sequence of actions to solve some problem?

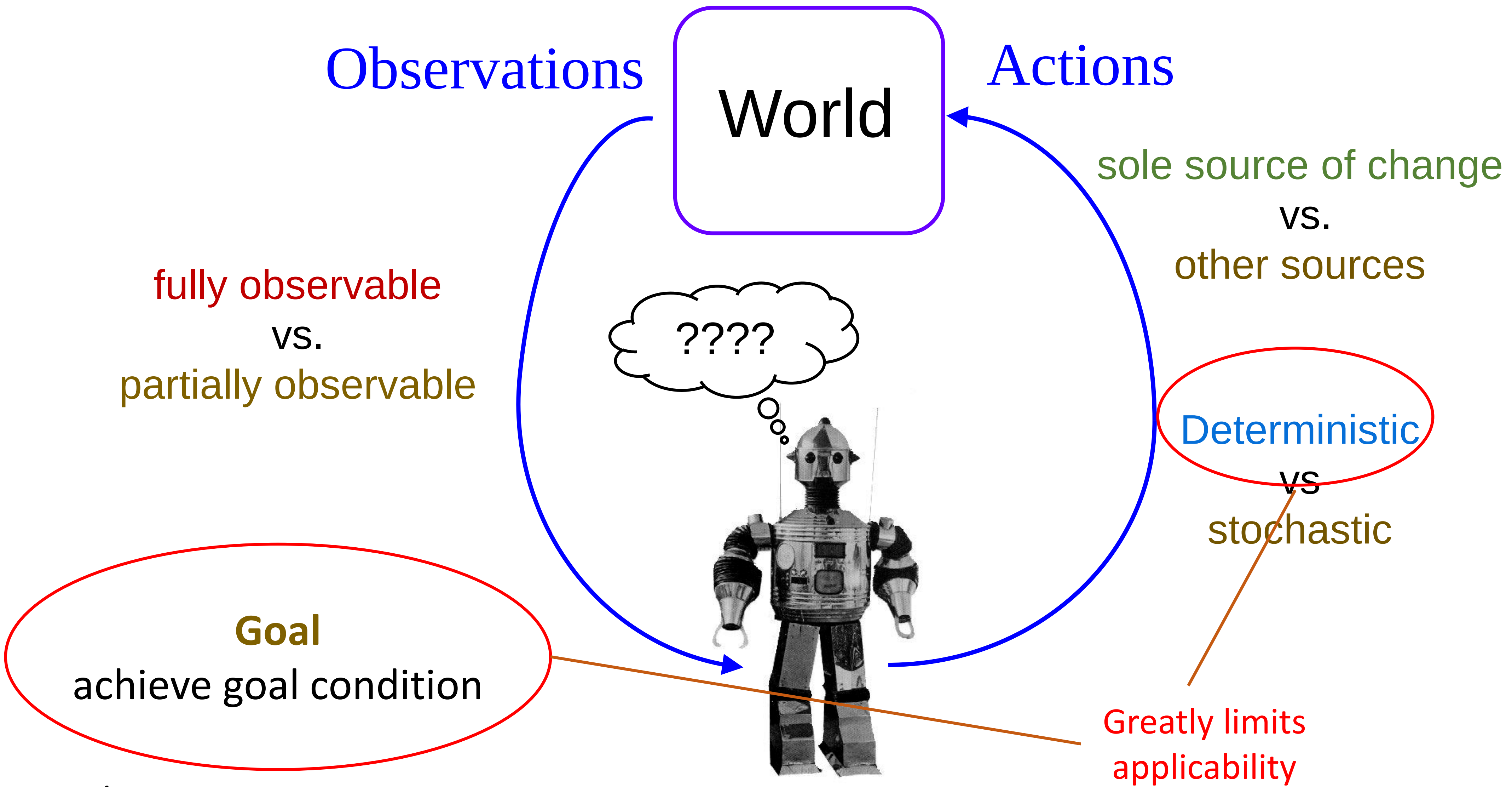
Autonomous vehicle navigation
Process control
Assembly line
Military operations
Travel planning
design and manufacturing environments
military operations, games, space exploration

- Military operations
- Autonomous space operations
- Construction tasks
- Machining tasks
- Mechanical assembly
- Design of experiments in genetics
- Command sequences for satellite

Some Dimensions of Planning



Classical Planning



Classical Planning

- Describe the world suing logic
- Given
  - a logical description of the **world states**
  - a logical description of a set of **possible actions**
  - a logical description of the **initial situation**
  - a logical description of the **goal conditions**
- Find
  - a **sequence of actions** that brings the agent from the initial situation to a situation in which the goal conditions hold.

<u>Classical planning:</u>
Environment
–Fully observable
–Deterministic
–Static
–Discrete

Example: A Robot that Makes Tea

1. put water in the kettle
2. heat the kettle
3. get a cup
4. pour hot water into the cup (after the water is hot enough)
5. get a tea bag
6. leave the tea bag in the water for enough time
7. remove the tea bag
8. add milk
9. add sugar
10. stir until mixed

Example: A Robot that Makes Tea

1. put water in the kettle
2. heat the kettle
3. get a cup
4. pour hot water into the cup (after the water is hot enough)
5. get a tea bag
6. leave the tea bag in the water for enough time
7. remove the tea bag
8. add milk
9. add sugar
10. stir until mixed

Sub-actions, are often needed

- "get a cup" might consist of the actions
  A. Move to the cupboard
  B. Grasp a cup
  C. Take it out of the cupboard
- Any of these actions could further broken down into smaller actions
  - The exact finger movements needed to grasp a cup
- **Exceptional** situations might trigger entire **sub-plans**
  - E.g. if there's no milk, the "get milk" action might trigger one of the following sub-plans
    - Go to the store to buy milk
- Actions might also **fail**, e.g. the cup could slip and fall to the floor
  - Must fix, or re-do, failed actions

Example: A Robot that Makes Tea

1. put water in the kettle
2. heat the kettle
3. get a cup
4. pour hot water into the cup (after the water is hot enough)
5. get a tea bag
6. leave the tea bag in the water for enough time
7. remove the tea bag
8. add milk
9. add sugar
10. stir until mixed

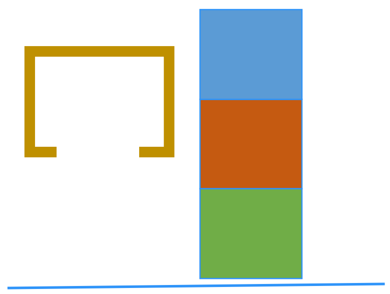
- **Timing** and **sensing** are also important
  - The kettle can't be heated for too short a time or too long a time.
  - The robot might need to taste the tea to decide if more milk/sugar is needed.
  - How does the robot know when the mixing is done?

Example: Dialog Planning

- Supermarket conversation:
  - Customer: Can you tell me where I can find the bread?
  - Employee: It’s in aisle 2.
  - Customer: Thanks!
- A dialog like this can be modelled as a planning problem
  - The customer’s goal is to get bread
  - He could do that in various ways, e.g. walking around the store searching for bread, finding a map that says where bread is, asking someone for help, etc.
  - In this dialog, the customer has chosen to try the “ask someone for help” action

Represent this Blocks World

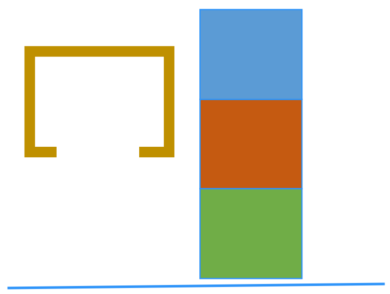
- A robot arm (yellow) can pick up and put down blocks to form stacks.
- It cannot pick up a block that has another block on top of it.
- It cannot pick up more than one block at a time.
- Any number of blocks can sit on the table.



- How would you represent this block world to use logic to find a plan?
- You need to represent the states, actions, goals, transitions.

Represent this Blocks World

- A robot arm (yellow) can pick up and put down blocks to form stacks.
- It cannot pick up a block that has another block on top of it.
- It cannot pick up more than one block at a time.
- Any number of blocks can sit on the table.



- How would you represent this block world to use logic to find a plan?
- You need to represent the states, actions, goals, transitions.

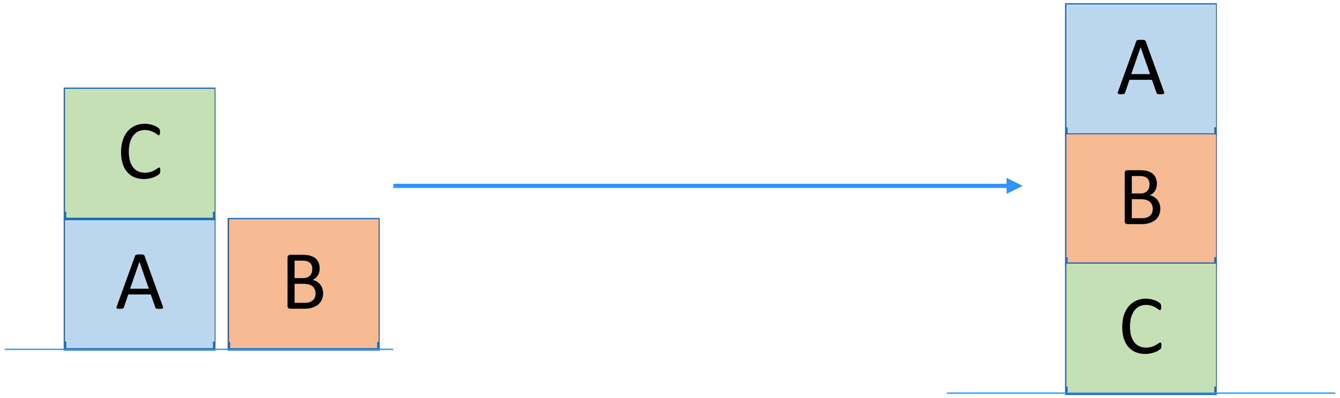
Classical Planning:

- \+ concise object representation and clear action definitions
- – only works for deterministic fully observable worlds

Blocks World

Task: Find a (short) sequence of block moves that transforms the starting state into the goal state.

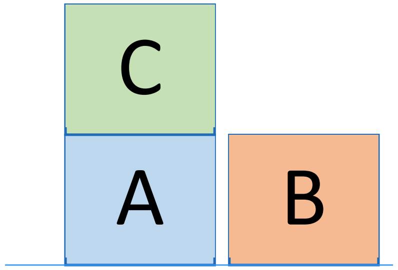
- A robot arm can pick up and put down blocks to form stacks.
- It cannot pick up a block that has another block on top of it.
- It cannot pick up more than one block at a time.
- Any number of blocks can sit on the table.



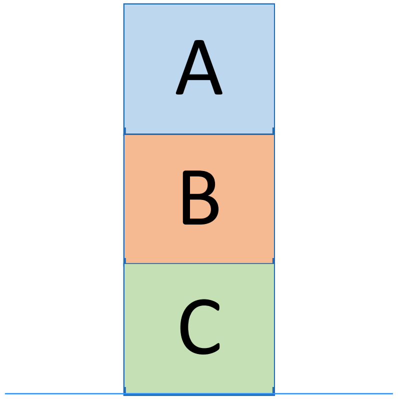
Starting state

Goal state

Predicates for the Blocks World



Starting state



Goal state

- A robot arm can pick up and put down blocks to form stacks.
- It cannot pick up a block that has another block on top of it.
- It cannot pick up more than one block at a time.
- Any number of blocks can sit on the table.

Predicates describing the initial state:

On(C, A), On(A, Table), On(B, Table), Clear(C), Clear(B)

Predicates describing the target state: On(A, B), On(B, C)

Move(X, Y) Precond: Clear(X), Clear(Y) Effect: On(X, Y)

Move(X, Table) Precond: Clear(X) Effect: On(X, Table)

Languages for Planning Problems

- STRIPS (Fikes and Nilsson, 1971)
  - Stanford Research Institute Problem Solver
  - Represent the world using a KB of first-order logic
  - Actions can change what is currently true
- PDDL
  - Planning Domain Definition Language
  - Revised & enhanced for the needs of the International Planning Competition

STRIPS language

State of the world = conjunction of positive, ground, function-free literals

At(Home) AND IsAt(Umbrella, Home) AND CanBeCarried(Umbrella) AND IsUmbrella(Umbrella) AND HandEmpty AND Dry

- Any literal not mentioned is assumed false

Action Definition:

1. a set PRE of preconditions facts
2. a set ADD of add effect facts
3. a set DEL of delete effect facts

PutDown(A,B):

1. PRE: { holding(A), clear(B) }
2. ADD: { on(A,B), handEmpty, clear(A) }
3. DEL: { holding(A), clear(B) }

PDDL

Planning Domain Definition Language

- Preconditions and goals can contain negative literals

Action Representation

- Action Schema
  - Action name
  - Preconditions
  - Effects
- Example

At(WHI,LNK),Plane(WHI),
Airport(LNK), Airport(OHA)

Fly(WHI,LNK,OHA)

At(WHI,OHA), ¬ At(WHI,LNK)

Action(Fly(p,from,to),

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

EFFECT: ¬At(p,from) ∧ At(p,to))

- Sometimes, Effects are split into ADD list and DELETE list

Action TakeObject

TakeObject(location, x)

Preconditions:

- HandEmpty
- CanBeCarried(x)
- At(location)
- IsAt(x, location)

Effects

- Holding(x)
- NOT(HandEmpty)
- NOT(IsAt(x, location))

WalkWithUmbrella (location1, location2, umbr)

- Preconditions:
  - At(location1)
  - Holding(umbr)
  - IsUmbrella(umbr)
- Effects:
  - At(location2)
  - NOT(At(location1))

WalkWithoutUmbrella (location1, location2)

- Preconditions:
  - At(location1)
- Effects:
  - At(location2)
  - NOT(At(location1))
  - NOT(Dry)

Goal: At(Work) AND Dry

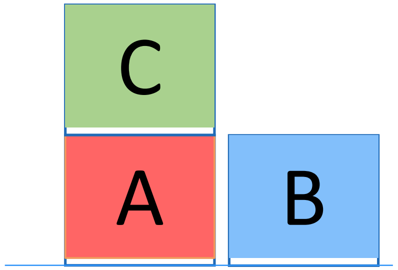
Initial state:

- At(Home) AND IsAt(Umbrella, Home) AND CanBeCarried(Umbrella) AND IsUmbrella(Umbrella) AND HandEmpty AND Dry

- TakeObject(Home, Umbrella)
  - At(Home) AND CanBeCarried(Umbrella) AND IsUmbrella(Umbrella) AND Dry AND Holding(Umbrella)
- WalkWithUmbrella(Home, Work, Umbrella)
  - At(Work) AND CanBeCarried(Umbrella) AND IsUmbrella(Umbrella) AND Dry AND Holding(Umbrella

Blocks World Example

Input Representation

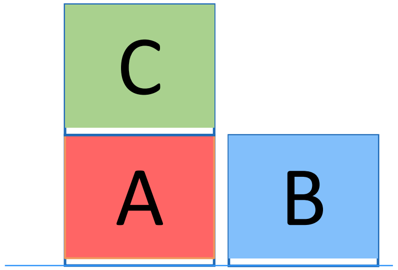


block (a), block (b), block (c),
on-table (a), on-table(b),
clear (a), clear (b), clear (c), arm-empty()

Generalize with variables

- **block(x)** means object x is a block
  - the table is an object, but not a block
- **on(x, y)** means object x is on top of object y
  - on(x,x) is not allowed, i.e. an object can’t be on top of itself
  - on(x,y) and on(y,x) cannot *both* be true at the same time
- **clear(x)** means there is nothing on top of object x
  - without this, we’d have to use quantified statements like “for all blocks y, on(y,x) is false

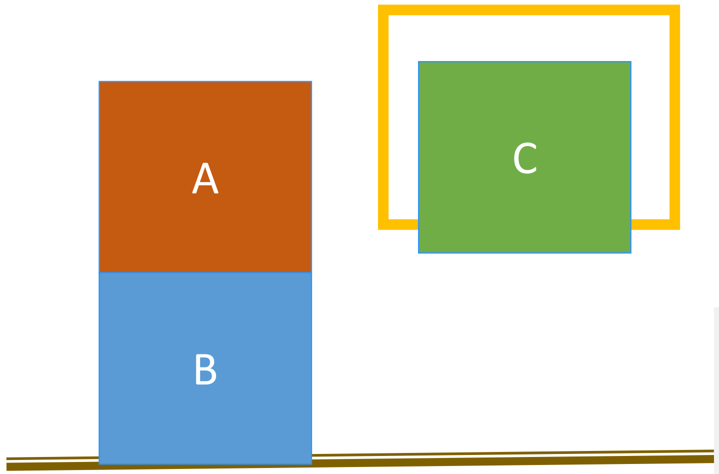
Blocks World Representation



Representation of this State

- block(A), block(B), block(C)
- on(C, A)
- on-table (A), on-table(B)
- clear(C), clear(B)

How to represent this state?



- block(A), block(B), block(C)
- on-block (A, B)
- on-table(B)
- clear(A)
- In-hand(C)

Goal Description

Description of goal: i.e. set of worlds

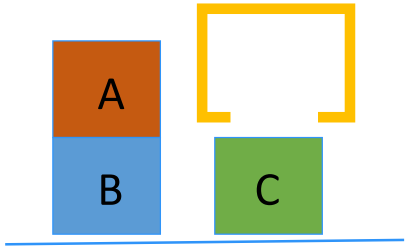
- E.g., Logical conjunction
- Any world satisfying conjunction is a goal

and (on-block (a, b) , on-block (b ,c))

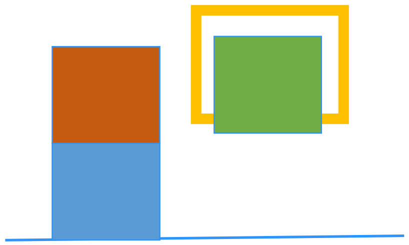
Pickup Block C from Table (Preconditions, Effects)

Instances:
Blocks A, B, C

Possible Predicates:
HandEmpty()
On-Table(block)
On-Block(b1,b2)
Clear(block)
In-Hand(block)

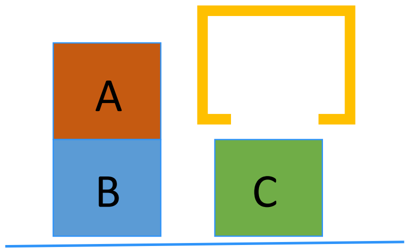


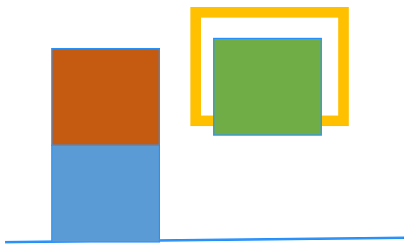
State:
HandEmpty()
On-Table(B)
On-Table(C)
On-Block(A,B)
Clear(A)
Clear(C)



State:
In-Hand(C)
On-Table(B)
On-Block(A,B)
Clear(A)
Clear(C)
Delete HandEmpty()
Delete On-Table(C)

Operator: Pickup-Block-C from Table





Preconditions	Effects
HandEmpty()	Add In-Hand(C)
Clear(C)	Delete HandEmpty()
On-Table(C)	On-Table(C)

Operators for Block Stacking

Pickup_Table(b):

Pre: HandEmpty(), Clear(b), On-Table(b)

Add: In-Hand(b)

Delete: HandEmpty(), On-Table(b)

Putdown_Table(b):

Pre: In-Hand(b)

Add: HandEmpty(), On-Table(b)

Delete: In-Hand(b)

Pickup_Block(b,c):

Pre: HandEmpty(), On-Block(b,c), b!=c

Add: In-Hand(b), Clear(c)

Delete: HandEmpty(), On-Block(b,c)

Putdown_Block(b,c):

Pre: In-Hand(b), Clear(c)

Add: HandEmpty(), On-Block(b,c)

Delete: Clear(c), In-Hand(b)

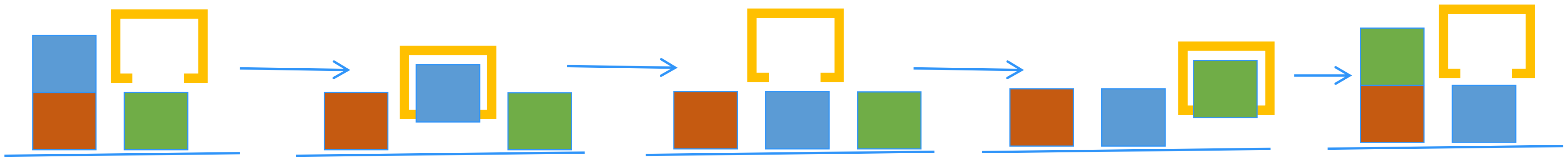
Why do we need separate operators for table vs on a block?

Example Matching Operators

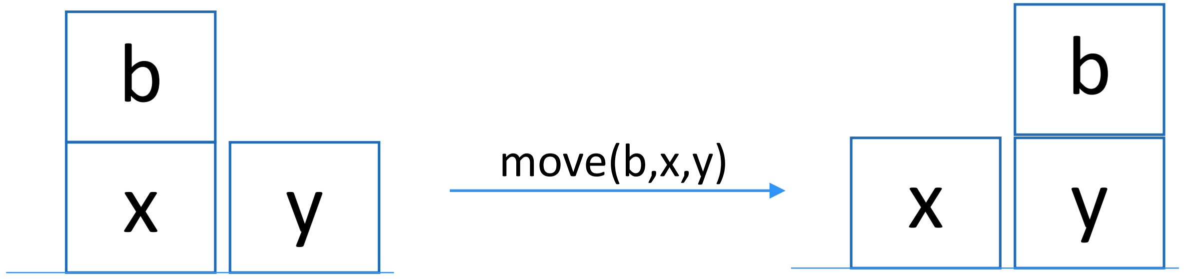
HandEmpty() & On-Table(O) & On-Block(B,O) & Clear(B) & On-Table(G) & Clear(G)

Pickup_Block(b,c):
Pre: HandEmpty(), On-Block(b,c), b!=c
Add: In-Hand(b), Clear(c)
Delete: HandEmpty(), On(b,c)

Pickup_Table(b):
Pre: HandEmpty, Clear(b), On-Table(b)
Add: In-Hand(b)
Delete: HandEmpty(), On-Table(b)



Blocks World: Alternative Action Schemas



Pre-condition: Facts that must be true before the action can be applied.

Name

Move block b from x to y.

Move(b, x, y)
Pre-condition: on-block (b,x) & clear(b) & clear(y)
Effect: on(b,y) & clear(x) & !on(b,x) & !clear(y)

Effect: What facts have changed after the action has been applied to a state.

This is called a **schema** because b, x, and y are variables, and for a concrete action they must each be assigned to an object.

Notice that we must explicitly say that on(b,x) and clear(y) are no longer true.