# Modeling Planning Problems in PDDL

Artificial Intelligence 2024-2025

# Contact Information

Dr. Davide Mario Longo

Cubo 41C, 3rd floor

Email: davidemario.longo@unical.it

# What is Planning?

# What is AI Planning?

**Planning as model-based goal-directed behavior:**

- Study of computational methods to **formulate and execute a plan** that achieves a **goal** starting from a known **initial state**
- Planners **reason about the future** by using a **model of the world** to evaluate the consequences of actions
- Planning is closely linked to **knowledge representation** and **problem solving**, bridging how we model a task and how we find a solution.

# Key Elements of Planning

**Planning system = Domain + Problem → Plan**

- **Domain**: defines available **actions** and relevant **state variables**
- **Problem**: specifies **initial state** and the **goal condition**
- The **planner** uses these to produce a **plan**, i.e., a sequence of actions

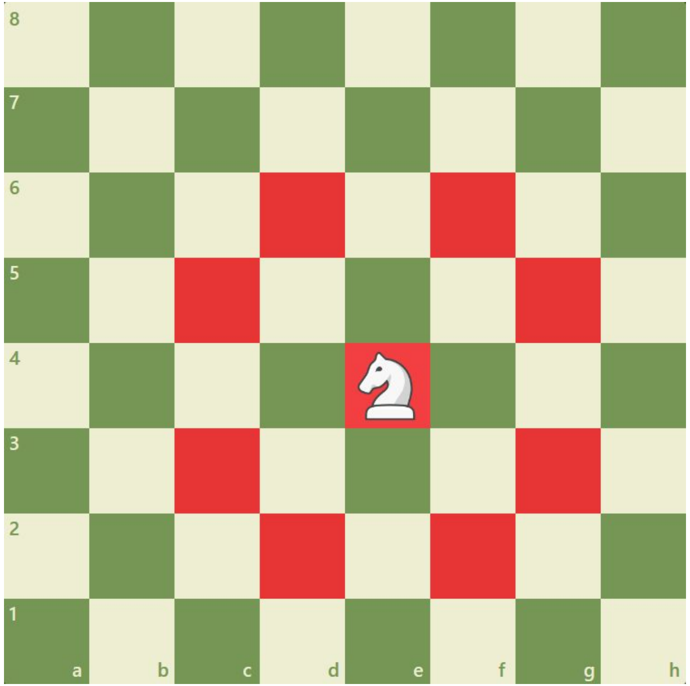
# Planning Example: Knight’s Tour

Problem Overview:

- Chess puzzle
- **Goal**: Move knight to visit every square exactly once
- Special case of Hamiltonian Path

Knight’s Move:

- Moves in an L-shape
- “Jumps” over intermediate squares



# Planning Example: Logistics

Problem Overview:

- Deliver goods from central depot
- Customers have specific delivery requests
- Trucks must be scheduled and routed

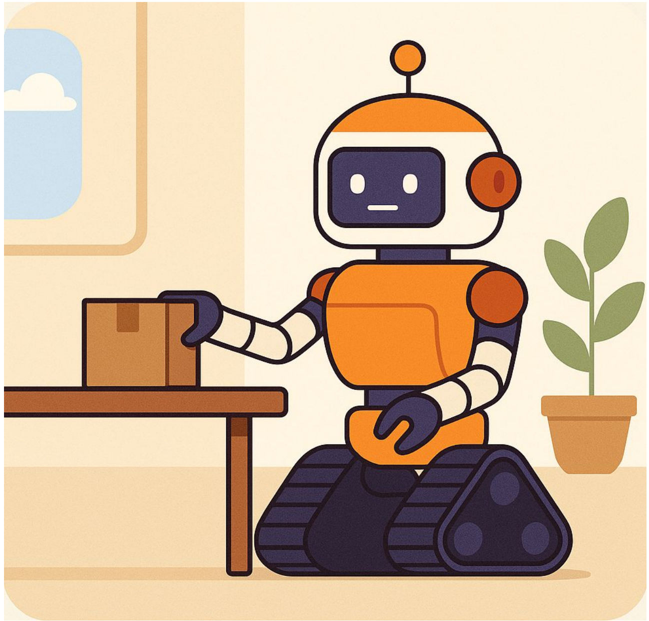
Key Challenges:

- Trucks differ in cost and capacity
- Only one trip per truck per day
- Minimize total delivery cost

# Planning Example: Robotics

Problem Overview:

- Control robot for goal-directed tasks
- Physical, dynamic environment
- Tasks: navigation, manipulation, sensing



Key Challenges:

- Plan under uncertainty
- Discrete actions + continuous motion
- Modeling preconditions and effects of robot actions

# Planning Example: Storytelling

What Is It?

- Automatically generate coherent, causal stories
- Characters act based on motivations and goals
- Stories unfold as sequences of planned actions

Key Challenges

- Ensure causal and emotional coherence
- Handle multiple agents with conflicting goals
- Blend logical planning with narrative structure

# Domain-Independent Planning

- **Domain-independence**: Planners don't need to “understand” the domain—they work with any problem expressible in the modeling language
- **PDDL** (Planning Domain Definition Language): the standard way of specifying such problems

# PDDL Basics

# What is PDDL?

Key Points:

- **PDDL (Planning Domain Definition Language)** is the **de-facto standard** for modeling planning problems
- Developed for the **International Planning Competition (IPC)** to standardize input formats
- Allows for **domain-independent planning**, where a planner can be reused across problems by simply changing the model input

# Key Features of Classical Planning in PDDL

PDDL splits definitions into two parts:

- **Domain file**: Defines the structure of the world (types, predicates, actions)
- **Problem file**: Defines a specific instance (objects, initial state, goal

They are typically stored in two `.pddl` files.

# PDDL Domain File

**Domain File** - Defines the structure of the world**:**

- **Types** - categories of objects
- **Predicates** - define facts about the world
- **Actions** – define how the world can change:
    - **Preconditions** – what must be true to apply the action
    - **Effects** – what becomes true or false after applying it

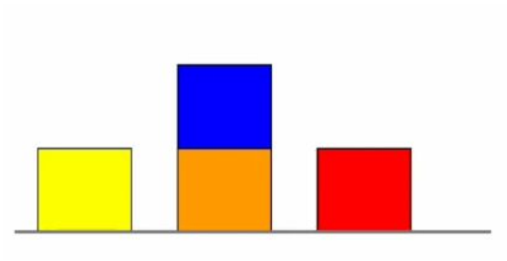
# PDDL Problem File

**Problem file** - Defines a specific instance:

- Objects - the actual things in the scenario
- Initial State - which facts are true at the beginning
- Goal - what facts should be true at the end of the plan

# Blocksworld Planning

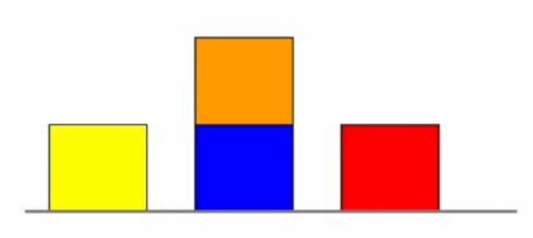
**Initial State**



**Plan**

1. **Unstack** the **blue** block from the **orange** block
2. **Put down** the **blue** block
3. **Pick up** the **orange** block
4. **Stack** the **orange** block on the **blue** block

**Goal State**



**Valid Actions**

- **Pick up** a block
- **Unstack** a block from another block
- **Put down** a block
- **Stack** a block on top of another block

**Constraints**

- I can only pick up one block at a time
- I can only pick up a block if my hand is empty
- I can only pick up a block if the block is on the table and the block is clear
- etc…

# Domain File Structure

- `(define (domain mydomain))`

Sections:

- `:requirements`
- `:types` (optional)
- `:predicates`
- `:action` definitions

**Note**: Order matters in readability, not functionality

# Syntax Basics

- Lisp-like syntax (S-expressions)
- Case-sensitive, mostly lowercase
- Variables start with ? (e.g., ?x)
- Comments: ; begins a comment line

# Blocksworld Domain File (1)

```
(define (domain blocksworld)
  (:requirements :strips :typing)

  (:types block)

  (:predicates
    (on ?x - block ?y - block)
    (ontable ?x - block)
    (clear ?x - block)
    (holding ?x - block)
    (handempty)
  )
…
)
```

# Problem File Structure

- `(define (problem myproblem))`
- Links to domain with `domain: mydomain`

Sections:

- `:objects` – constants in the problem
- `:init` – initial facts (true predicates)
- `:goal` – goal condition(s)

**Note**: Different problems can use the same domain

# Blocksworld Problem File (1)

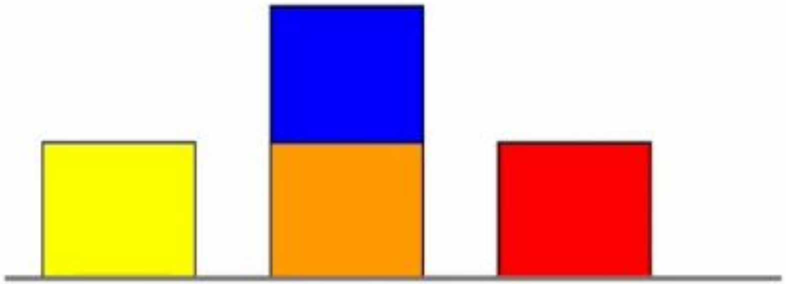
```
(define (problem blocksworld-example)
  (:domain blocksworld)


  (:objects
    red yellow blue orange - block
  )
  …
)
```

# Blocksworld Problem File (2)

```
(:init
   (ontable yellow)
   (ontable orange)
   (ontable red)
   (on blue orange)
   (clear blue)
   (clear red)
   (clear yellow)
   (handempty)
 )
```

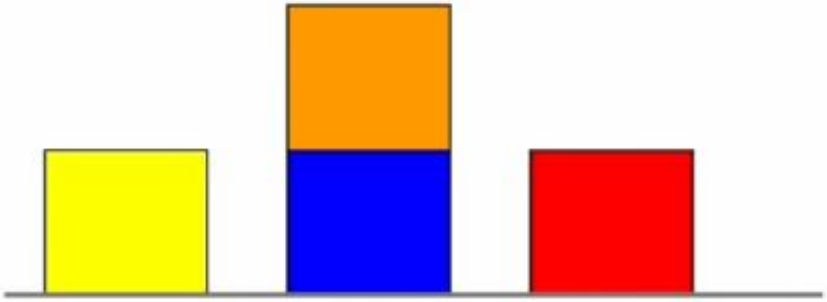
**Initial State**



# Blocksworld Problem File (3)

```
(:goal
    (and
      (on orange blue)
      (ontable yellow)
      (ontable blue)
      (ontable red)
      (clear orange)
      (clear yellow)
      (clear red)
    )
  )
```

**Goal State**



# Defining Actions in PDDL

# Structure of an Action

Action definition format:

```
(:action action-name
 :parameters (...)
 :precondition (...)
 :effect (...))
```

Each action describes:

- What it needs to happen (precondition)
- What it changes in the world (effect)

**Example Action Names**: `pick-up`, `put-down`, `unstack`, `stack`

# Parameters

- Variables used in the action
- Must be typed (if `:typing` is used)

Example:

```
:parameters (?x - block ?y - block)
```

Used to define general actions like:

- Unstacking block ?x from block ?y
- Stacking block ?x onto block ?y

Parameters get grounded with real blocks like `blue`, `orange`, etc.

# Preconditions

- Conditions that must be true before the action can execute
- Usually in `:precondition (and ...)`

Example:

```
:precondition (and (on ?x ?y) (clear ?x) (handempty))
```

To unstack block `?x` from `?y`:

- `?x` must be **on** `?y`
- `?x` must be **clear**
- Hand must be **empty**

These ensure logical actions (can’t grab a block under another!)

# Effects

- Describes how the world changes when the action is applied
- `:effect (and ...)` contains:
    - Add effects (facts that become true)
    - Delete effects (with `(not ...)`)

Example:

```
:effect (and
  (holding ?x) (clear ?y)
  (not (on ?x ?y)) (not (clear ?x)) (not (handempty))
)
```

**Standard semantics: Add after Delete**

# Full Action Example

```
(:action unstack
 :parameters (?x - block ?y - block)
 :precondition (and (on ?x ?y) (clear ?x) (handempty))
 :effect (and
   (holding ?x) (clear ?y)
   (not (on ?x ?y)) (not (clear ?x)) (not (handempty))
 )
)
```

# Your Turn

```
(:action pick-up
  :parameters
```

# Your Turn

```
(:action pick-up
  :parameters (?x - block)
  :precondition
```

# Your Turn

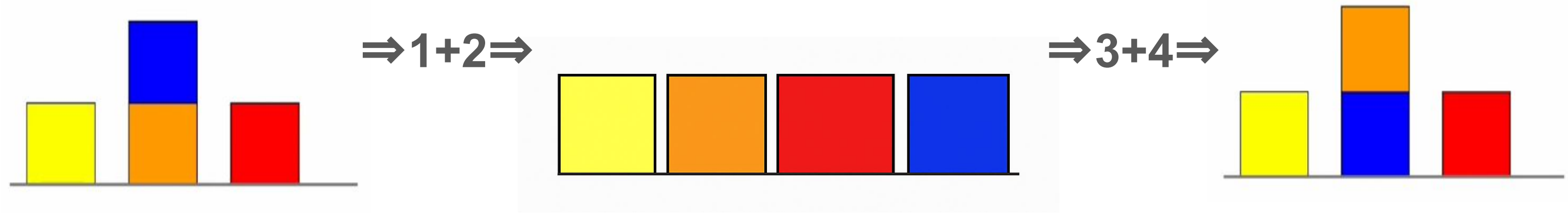
```
(:action pick-up
  :parameters (?x - block)
  :precondition (and (clear ?x) (ontable ?x) (handempty))
  :effect
```

# Your Turn (Solution)

```
(:action pick-up
  :parameters (?x - block)
  :precondition (and (clear ?x) (ontable ?x) (handempty))
  :effect (and (not (ontable ?x)) (not (clear ?x)) (not
(handempty)) (holding ?x))
)
```

# Plan Execution

1. `(unstack blue orange)`
2. `(put-down blue)`
3. `(pick-up orange)`
4. `(stack orange blue)`




The planner searches for a valid sequence that satisfies the goal

# What the Planner Does

**Input**: Domain file + Problem file

**Process**:

- Search for a sequence of valid actions
- Check preconditions/effects at each step
- Use STRIPS logic to simulate world changes

**Output**:

**A valid plan**: sequence of grounded actions that transitions from the **initial state** to the **goal state**

# Advanced Notes & Wrap-Up

# STRIPS Assumptions

PDDL’s basic structure is based on STRIPS:

- Deterministic actions
- Fully observable environment
- Discrete states

Limitations:

- No conditional effects
- No negative preconditions (in basic STRIPS)
- No time/duration or uncertainty

# Optional Features / Extensions

- **Typing**: cleaner models, constraints on parameters
- **Negative preconditions**: (not (at ?r ?from)) in preconditions
- **Numeric fluents**: track resources or time
- **Temporal planning**: durations, deadlines (:durative-action)
- **Hierarchical Task Networks (HTN)**: for complex plans

Most planners don’t support everything—depends on tool

# Tools and Planners

Fast Downward (https://www.fast-downward.org/)

- A powerful and widely used classical planner for PDDL.

VAL (Plan Validator) (https://github.com/KCL-Planning/VAL)

- Tool to **validate** a plan given a domain/problem and a planner output.

# Online Editors & Interfaces

Planning.domains – Editor & Visualizer (https://editor.planning.domains/)

- Online editor for PDDL with syntax highlighting
- Can run planning problems using integrated planners
- Supports visualizing state transitions and plans

# Benchmark Domains

IPC Domains & Problems (International Planning Competition) (https://ipc2023-benchmarks.github.io/)

- Collection of standard planning benchmarks
- Useful for testing and learning various planning domains
- Includes domains like blocks world, logistics, rover, etc.

# Large Language Models for Planning Tasks

LLMs are being explored for:

- Generating PDDL from natural language descriptions
- Explaining planning problems and action models
- Interacting with planners via conversational interfaces

Example use cases:

- Sketching domain models quickly
- Translating human goals into formal representations
- Interactive debugging of planning domains

# Summary & Takeaways

- PDDL models *what* can happen, not *how*
- Domain describes the logic, problem sets the context
- A planner generates a valid sequence of actions to reach the goal

Next steps:

- Try creating your own domain/problem pair
- Run it with a simple planner
- Explore advanced PDDL features