K. J. Somaiya College of Engineering, Mumbai-77

Batch: B2	**Roll No.: 16010122221**
Experiment / assignment / tutorial No. 7	

Title: Solving planning problems using STRIPS or PDDL tools.

Objective: To write STRIPS scripts to solve planning problem and implement them using PDDL tools

Expected Outcome of Experiment:

Course Outcome	After successful completion of the course students should be able to
CO2	Analyze and solve problems for goal based agent architecture (searching and planning algorithms).

Books/ Journals/ Websites referred:

1. https://planning.wiki/, last retrieved on Feb 27,2025
2. https://editor.planning.domains/, last retrieved on Feb 27,2025
3. https://www.youtube.com/watch?v=EeQcCs9SnhU, last retrieved on Feb 27,2025
4. https://www.youtube.com/watch?v=FS95UjrICy0, last retrieved on Feb 27,2025
5. https://nms.kcl.ac.uk/planning/software/optic.html, last retrieved on Feb 27,2025
6. https://github.com/yarox/pddl-examples, last retrieved on Feb 27,2025
7. https://planning.wiki/_citedpapers/pddl3bnf.pdf , last retrieved on Feb 27,2025
8. https://www.youtube.com/watch?v=XW0z8Oik6G8
9.
10. https://github.com/potassco/pddl-instances, last retrieved on Feb 27,2025
11. "Artificial Intelligence: a Modern Approach" by Russell and Norving, Pearson education Publications
12. "Artificial Intelligence" By Rich and knight, Tata McGraw Hill Publications

Pre Lab/ Prior Concepts:
Goal based agents, searching, uninformed search, informed search

Historical Profile: *(Details about planning Vs Searching)*

New Concepts to be learned:

Representing problem as planning problem, ADL, STRIPS, Total order plan, partial order plan

Chosen Planning Problem:

The chosen planning problem involves delivery management for an agent. The goal is to move items from one location to another using an agent, ensuring that items are loaded, delivered, and dropped off in the correct order across multiple locations. The problem includes several actions such as moving the agent, picking up and delivering items, and ensuring that the agent performs these actions while respecting the constraints of connected locations and the availability of items at specific locations.

STRIPS/ADL Script for solving problem:

STRIPS Script:

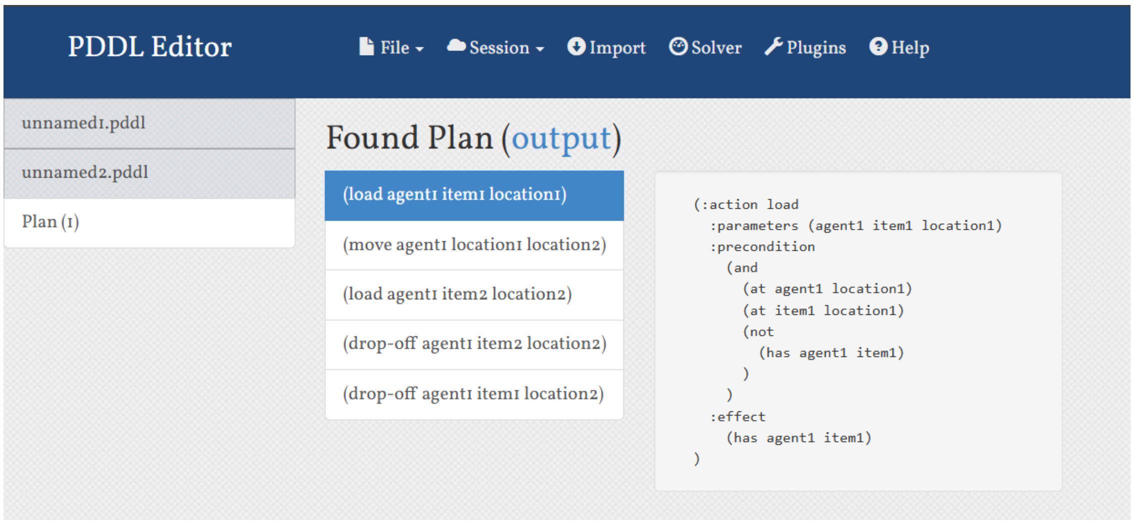
```
(define (domain delivery-management)
  (:requirements :strips :typing)
  (:types agent item location)
  (:predicates
    (at ?agent - agent ?loc - location)
    (at ?item - item ?loc - location)
    (has ?agent - agent ?item - item)
    (delivered ?item - item)
    (connected ?loc1 - location ?loc2 - location)
  )

  ;; Move action
  (:action move
    :parameters (?agent - agent ?from - location ?to - location)
    :precondition (and (at ?agent ?from) (connected ?from ?to))
    :effect (and (not (at ?agent ?from)) (at ?agent ?to))
  )

  ;; Pick-up action
  (:action pick-up
    :parameters (?agent - agent ?item - item ?loc - location)
    :precondition (and (at ?agent ?loc) (at ?item ?loc) (not (has ?agent ?item)))
    :effect (has ?agent ?item)
```

```
 )

 ;; Drop-off action
 (:action drop-off
   :parameters (?agent - agent ?item - item ?loc - location)
   :precondition (and (at ?agent ?loc) (has ?agent ?item))
   :effect (and (not (has ?agent ?item)) (delivered ?item))
 )

 ;; Load action
 (:action load
   :parameters (?agent - agent ?item - item ?loc - location)
   :precondition (and (at ?agent ?loc) (at ?item ?loc) (not (has ?agent ?item)))
   :effect (has ?agent ?item)
 )

 ;; Unload action
 (:action unload
   :parameters (?agent - agent ?item - item ?loc - location)
   :precondition (and (at ?agent ?loc) (has ?agent ?item))
   :effect (and (not (has ?agent ?item)) (delivered ?item))
 )

 ;; Pick-up with vehicle action
 (:action pick-up-with-vehicle
   :parameters (?agent - agent ?item - item ?loc - location)
   :precondition (and (at ?agent ?loc) (at ?item ?loc) (not (has ?agent ?item)))
   :effect (has ?agent ?item)
 )

 ;; Drop-off with vehicle action
 (:action drop-off-with-vehicle
   :parameters (?agent - agent ?item - item ?loc - location)
   :precondition (and (at ?agent ?loc) (has ?agent ?item))
   :effect (and (not (has ?agent ?item)) (delivered ?item))
 )

 ;; Move between locations with vehicle
 (:action move-with-vehicle
   :parameters (?agent - agent ?from - location ?to - location)
   :precondition (and (at ?agent ?from) (connected ?from ?to))
```

```
    :effect (and (not (at ?agent ?from)) (at ?agent ?to))
  )
)
```

PDDL Script for Solving Problem:

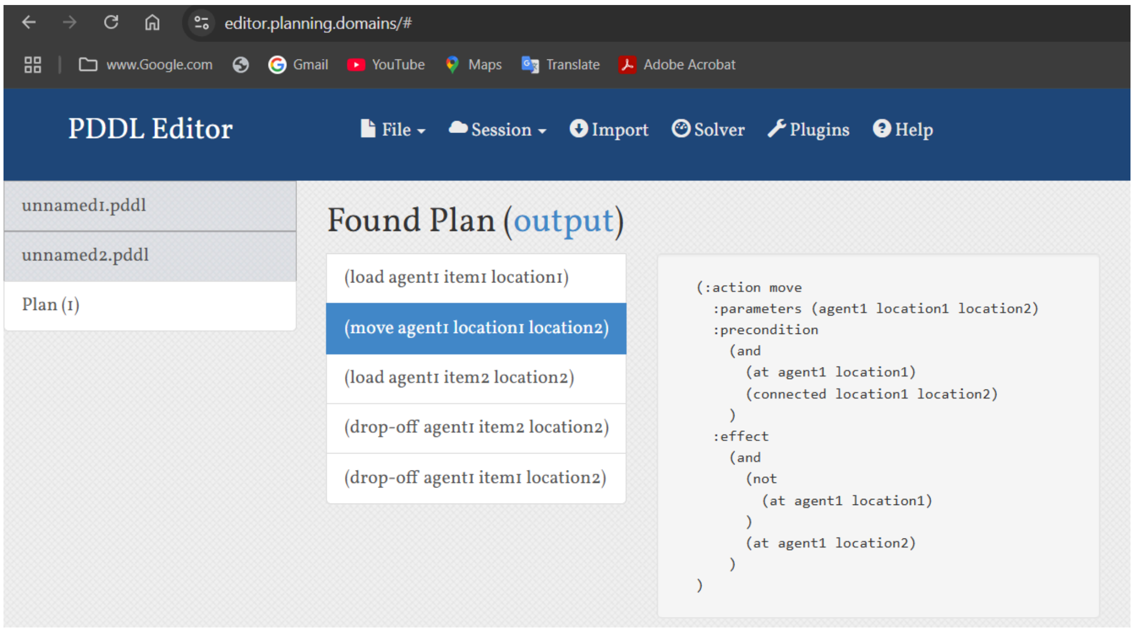
```
(define (problem delivery-problem)
  (:domain delivery-management)

  (:objects
    agent1 - agent
    item1 item2 - item
    location1 location2 location3 location4 - location
  )

  (:init
    (at agent1 location1)
    (at item1 location1)
    (at item2 location2)
    (connected location1 location2)
    (connected location2 location3)
    (connected location3 location4)
  )

  (:goal (and
    (delivered item1)
    (delivered item2)
  ))
)
```



Found Plan (output)

(load agent1 item1 location1)
(move agent1 location1 location2)
(load agent1 item2 location2)
(drop-off agent1 item2 location2)
(drop-off agent1 item1 location2)

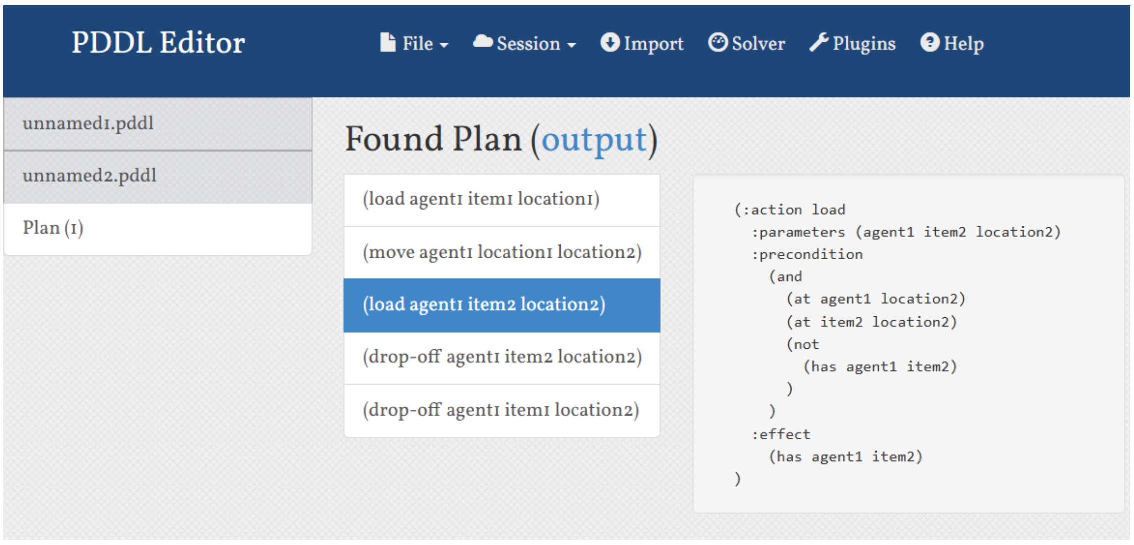
```
(:action load
  :parameters (agent1 item1 location1)
  :precondition
    (and
      (at agent1 location1)
      (at item1 location1)
      (not
        (has agent1 item1)
      )
    )
  :effect
    (has agent1 item1)
)
```



Found Plan (output)

(load agent1 item1 location1)
(move agent1 location1 location2)
(load agent1 item2 location2)
(drop-off agent1 item2 location2)
(drop-off agent1 item1 location2)

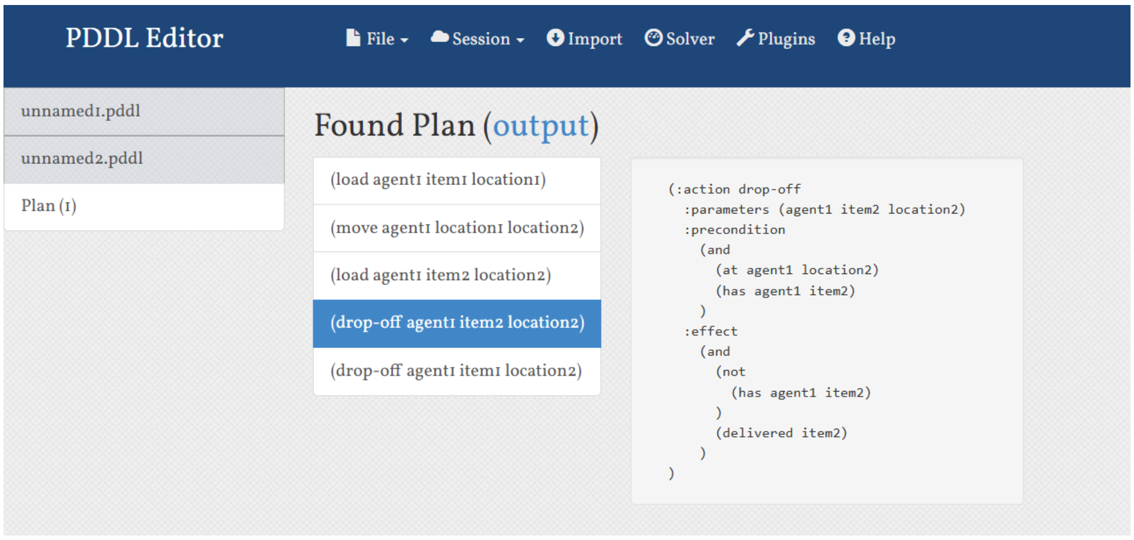
```
(:action move
  :parameters (agent1 location1 location2)
  :precondition
    (and
      (at agent1 location1)
      (connected location1 location2)
    )
  :effect
    (and
      (not
        (at agent1 location1)
      )
      (at agent1 location2)
    )
)
```



Found Plan (output)

(load agent1 item1 location1)
(move agent1 location1 location2)
(load agent1 item2 location2)
(drop-off agent1 item2 location2)
(drop-off agent1 item1 location2)

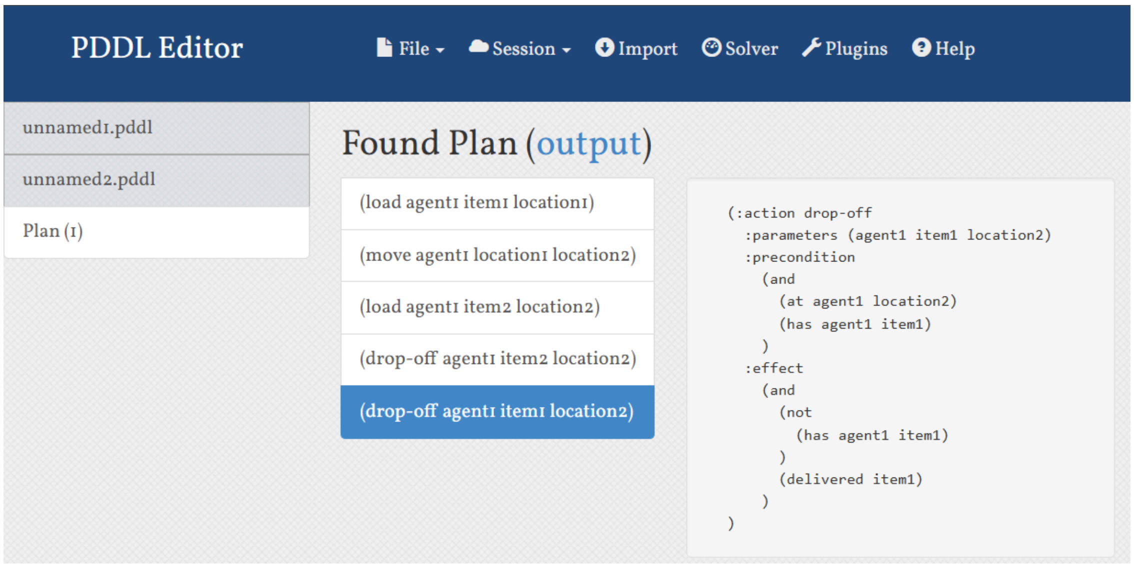
```
(:action load
  :parameters (agent1 item2 location2)
  :precondition
    (and
      (at agent1 location2)
      (at item2 location2)
      (not
        (has agent1 item2)
      )
    )
  :effect
    (has agent1 item2)
)
```



Found Plan (output)

(load agent1 item1 location1)
(move agent1 location1 location2)
(load agent1 item2 location2)
(drop-off agent1 item2 location2)
(drop-off agent1 item1 location2)

```
(:action drop-off
  :parameters (agent1 item2 location2)
  :precondition
    (and
      (at agent1 location2)
      (has agent1 item2)
    )
  :effect
    (and
      (not
        (has agent1 item2)
      )
      (delivered item2)
    )
)
```



Found Plan (output)

(load agent1 item1 location1)
(move agent1 location1 location2)
(load agent1 item2 location2)
(drop-off agent1 item2 location2)
(drop-off agent1 item1 location2)

```
(:action drop-off
  :parameters (agent1 item1 location2)
  :precondition
    (and
      (at agent1 location2)
      (has agent1 item1)
    )
  :effect
    (and
      (not
        (has agent1 item1)
      )
      (delivered item1)
    )
)
```

Explanation of PDDL Model:

The Planning Domain Definition Language (PDDL) model provides a structured way to define planning problems for automated planners. The model consists of two primary parts: the domain and the problem. The domain describes the general rules of action and the problem defines the specific instance of the planning task to be solved.

1. Domain Description

The domain defines the types of objects, predicates, and actions that are relevant to the planning problem. It acts as a blueprint or template for solving a variety of planning problems that share the same structure.

In this example, the delivery management domain involves actions like moving an agent between locations, picking up and dropping off items, and ensuring that items are delivered successfully. Below is a breakdown of key components of the domain:

Types:

Types define the categories of objects involved in the domain. In our problem, we have three types:

- agent: Represents the entity responsible for delivering the items.
- item: Represents the object being delivered.
- location: Represents the places where the agent can be and where the items are located.

Predicates:

Predicates define the properties or relationships between objects at any given time. They are used to describe the state of the world in which the agent operates. Predicates can be thought of as variables that represent facts about objects.

- (at ?agent ?loc): This predicate states that the agent is located at a specific location.
- (at ?item ?loc): This predicate indicates that a particular item is at a specific location.
- (has ?agent ?item): This predicate means that the agent has the item in its possession, i.e., it has picked it up.
- (delivered ?item): This predicate specifies that the item has been delivered successfully to the goal location.

- (connected ?loc1 ?loc2): This predicate indicates that there is a connection between two locations, meaning the agent can move between them.

Actions:

Actions define the operations that the agent can perform to change the state of the world. Each action consists of:

1. Parameters: The objects involved in the action (e.g., agent, item, location).
2. Preconditions: Conditions that must be true for the action to be executed.
3. Effects: The changes that occur in the world after the action is executed.

In this model, there are several actions:

- move: This action moves the agent from one location to another. The precondition is that the agent is at the starting location and that there is a connection between the two locations. The effect is that the agent is now at the destination location.
- pick-up: This action allows the agent to pick up an item. The precondition is that the agent is at the location where the item is located and the agent does not already have the item. The effect is that the agent now has the item in its possession.
- drop-off: This action enables the agent to drop off the item at its destination. The precondition is that the agent is at the location and the agent is holding the item. The effect is that the item is now considered delivered, and the agent no longer has the item.
- load: This action is similar to pick-up but might represent a more specific operation (e.g., the agent could be loading the item into a vehicle).
- unload: This action is similar to drop-off, representing the unloading of an item from a vehicle.
- pick-up-with-vehicle: This action represents picking up an item while using a vehicle, where the agent is at a location with the item and does not already have it.
- drop-off-with-vehicle: This action allows the agent to deliver an item to a location using a vehicle.

Each of these actions has a set of preconditions that must hold for the action to take place. For example:

- In the move action, the precondition is that the agent is at the current location and there is a valid connection between the two locations.
- In the pick-up action, the agent must be at the same location as the item and not already possess it.

The effects of actions specify how the world changes after the action is executed. For example:

- After move is executed, the agent is now at the destination, and the predicate `(at ?agent ?to)` is true.
- After pick-up, the agent holds the item, and the predicate `(has ?agent ?item)` becomes true.
- After drop-off, the item is delivered, and the predicate `(delivered ?item)` becomes true.

2. Problem Description

The problem specifies the initial state of the world and the goal that the planner needs to achieve. The initial state consists of the positions of the agent and items and the connections between locations. The goal defines the conditions that must be true in the final state for the problem to be solved.

In this problem, the initial state includes:

- The agent starts at a particular location (e.g., location1).
- Items (e.g., item1 and item2) are located at specific locations (e.g., item1 at location1, item2 at location2).
- There are connections between locations, such as location1 connected to location2, location2 connected to location3, and so on.

The goal is that both items should be delivered, i.e., the conditions `(delivered item1)` and `(delivered item2)` should be true.

3. Interaction Between Domain and Problem

The domain provides the template for the actions, predicates, and types, while the problem script provides specific details for the instance of the problem. For instance, the domain defines that an agent can move and pick up items, but the problem script provides the initial locations of the agent and items and specifies the goal (delivering the items).

The planner takes the domain and problem as input and uses the defined actions and predicates to generate a plan that achieves the goal.

4. How the Model Solves the Problem

The planner searches for a sequence of actions that will lead from the initial state to the goal state. In this case, the planner would:

1. Identify the agent’s initial location and the locations of the items.
2. Check the actions available, such as `move`, `pick-up`, and `drop-off`.
3. Generate a sequence of actions that will move the agent to the right locations, pick up the items, and deliver them to the destination locations.

The planner will use the preconditions of each action to determine whether it is possible to perform that action in the current state and will generate the plan by applying actions in a valid sequence.

Post Lab Descriptive Questions:

1. **How does ADL (Action Description Language) extend STRIPS?**
   ADL extends STRIPS by allowing more complex conditions and effects for actions. While STRIPS restricts the preconditions and effects to positive literals (e.g., "at(agent, location)"), ADL supports quantified expressions, disjunctions (logical OR), and negations in conditions and effects. This makes ADL more expressive, as it can handle a broader range of planning problems compared to STRIPS.
2. Define **Partial Order Planning (POP)** and **Total Order Planning (TOP)**.

   Partial Order Planning (POP): In POP, actions are ordered in a way that respects their dependencies but does not enforce a strict sequential order. As long as actions that depend on each other are ordered correctly, other actions can be executed independently. POP is more flexible as it allows multiple actions to be executed in parallel.

   Total Order Planning (TOP): In TOP, actions are ordered strictly from start to finish, meaning each action follows a specific sequence. This results in less flexibility compared to POP because all actions must be executed in a strict order, and parallelism is not utilized.

3. How do they differ?

   The main difference between POP and TOP is that POP allows more flexibility in the ordering of actions, potentially enabling parallel execution, whereas TOP enforces a sequential execution of actions.

4. Would **Partial Order Planning** be beneficial in this problem? Why or why not?

   Yes, Partial Order Planning (POP) would be beneficial in this problem. Since multiple actions (such as moving to locations, picking up items, and dropping them off) can potentially be performed in parallel or in a flexible sequence, POP allows the agent to avoid unnecessary sequential restrictions and optimizes the plan. This flexibility could lead to more efficient delivery planning, especially if there are multiple items or more locations involved.

5. Correlate the knowledge engineerings steps with PDDL scripts

   1. Problem Definition: The knowledge engineering process begins with defining the problem's domain and specific requirements. The PDDL domain script defines the general actions, conditions, and effects.

   2. Predicate Definition: In PDDL, predicates represent the state of the world. Knowledge engineering involves determining the relevant states that must be tracked (e.g., agent location, item location, item possession, delivery status).

   3. Action Specification: The actions available to the agent (such as moving, picking up, or delivering items) are described in the PDDL action sections.

These actions are based on the planning problem's requirements and constraints.

4. Goal State: The goal is defined in the PDDL goal section, stating what conditions must be true for the plan to be successful. This correlates with the problem's desired outcome (items being delivered).

5. Problem Formulation: The PDDL problem script represents the specific instance of the planning problem, including objects, initial conditions, and the goal. This corresponds to setting up the specific details of the scenario being solved (e.g., locations, items, agent).

Conclusion

In this PDDL model, the problem and domain are clearly defined to allow the planner to determine an optimal sequence of actions that will move the agent, pick up items, and drop them off at the correct locations. The domain provides the framework for the types of actions available, while the problem defines the specific instance of the task with initial conditions and a goal.