CS 253

**REPORT**

CLASSICAL PLANNING

ARYAN MAURYA

horizontal line

# 

# Introduction

PDDL (Planning Domain Definition Language) is a formal language used in Artificial Intelligence and automated planning. It is used to describe the essential components of a planning problem, including the initial state of the problem, the actions available to an agent, and the goal state that the agent is trying to achieve. PDDL provides a way to model the world in terms of objects, their properties, and the relationships between them. It also allows for the specification of preconditions and effects of actions, which are used to define how the world changes as a result of an agent's actions. PDDL is an expressive language that can be used to describe a wide range of planning problems, from simple toy problems to complex real-world scenarios. It has been used extensively in the field of automated planning, both as a means of describing planning problems and as a way of evaluating planning algorithms and systems. Overall, PDDL is an important tool in the development and evaluation of automated planning systems, and it continues to be an active area of research in the field of Artificial Intelligence.

## PDDL SYNTAX

The syntax of PDDL (Planning Domain Definition Language) is designed to be simple and easy to understand. Here is an overview of the main elements of the language:

1. Domain definition: This section defines the types of objects that can exist in the problem domain, as well as the actions that can be performed on those objects.

### 

(define (domain domain\_name)

(:requirements requirement\_name1 requirement\_name2 ...)

(:types type\_name1 type\_name2 ...)

(:predicates predicate1 predicate2 ...)

(:action action\_name

:parameters (?parameter1 ?parameter2 ...)

:precondition (precondition\_formula)

:effect (effect\_formula))

...)

1. **Problem definition:** This section defines the initial state of the problem, as well as the goal that the planner is trying to achieve.

(define (problem problem\_name)

(:domain domain\_name)

(:objects object1 object2 ...)

(:init (predicate1) (predicate2) ...)

(:goal (goal\_formula)))

In these definitions, domain\_name and problem\_name are strings that identify the domain and problem, respectively. requirement\_name specifies any additional requirements needed to define the domain, such as whether or not negative preconditions are allowed.

type\_name defines the types of objects that exist in the domain, and predicate defines the properties or relations between those objects. action\_name defines a specific action that can be performed on objects in the domain.

parameter specifies the objects that are involved in the action, and

precondition\_formula and effect\_formula specify the conditions that must be true before the action can be performed, and the changes that occur as a result of the action, respectively.

object defines the specific instances of objects in the problem,

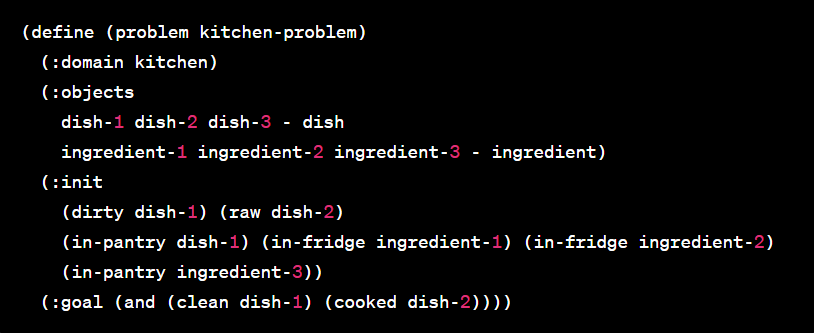
predicate defines the properties or relations of the objects in the initial state,

and goal\_formula defines the desired properties or relations of the objects in the goal state.

# EXAMPLE

## DOMAIN FILE



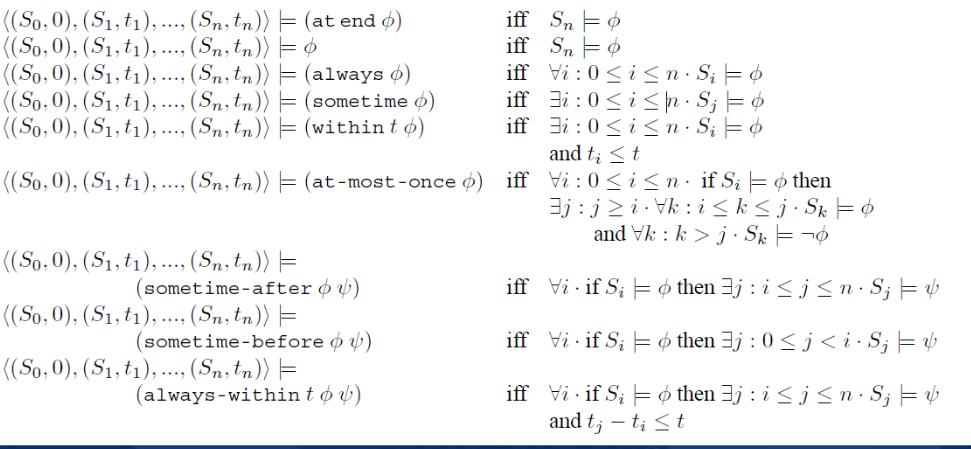
**PROBLEM FILE-**

The problem statement for the above PDDL domain and problem files is to clean dish-1 and cook dish-2 in the context of a kitchen.

The problem statement is described in the problem file, where the initial state of the world is defined using the :init keyword. Here, dish-1 is dirty and dish-2 is raw, and several other objects are in their default positions.

The goal of the problem is defined in the :goal section, which specifies that the planner should achieve two conditions: clean dish-1 and cooked dish-2. The planner must use the actions defined in the domain file to transform the initial state into the goal state.

# TEMPORAL CONSTRUCT



## CONSTRAINTS IN PDDL

**1)at end**

**2)always** - The always state-trajectory constraint expresses that every state reached in the execution of the plan, contains the predicate specified.

**3)sometime-** The sometime state trajectory constraint expresses at some point within the states reached by a plan, that the predicate specified is true.

**4)within -** The within state-trajectory constraint express that some predicate must become true within the specified number of plan steps.

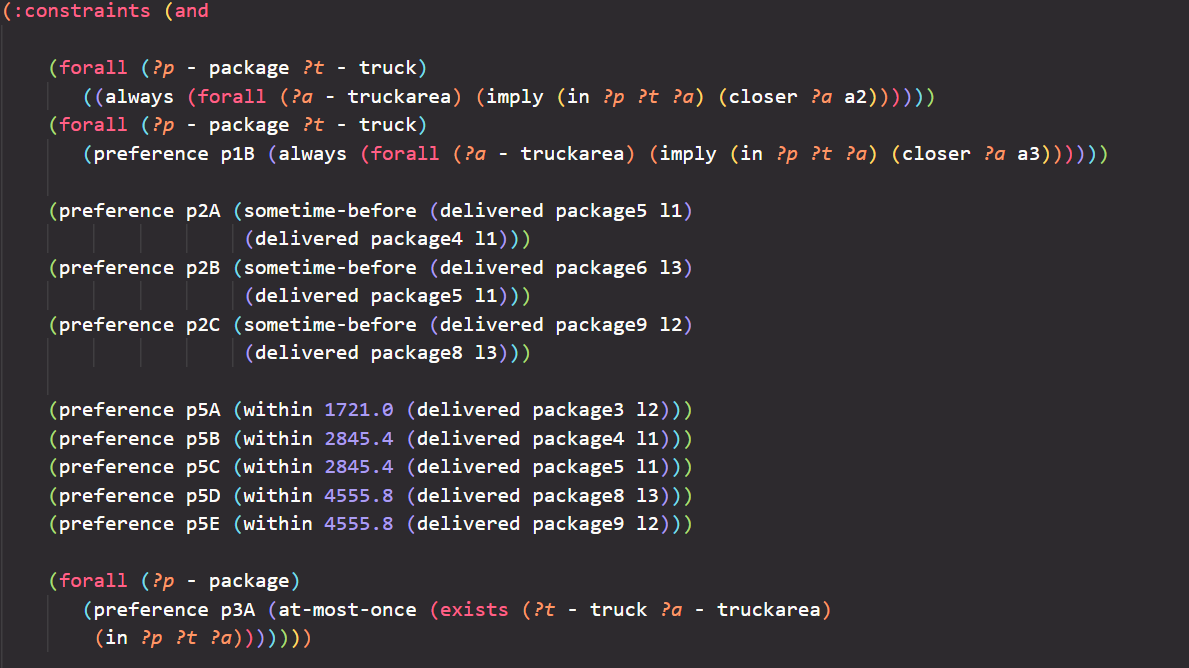
**5)At-most-once** - The at-most-once state-trajectory constraint expresses that a fact be true at most once. It is useful to prevent repeated visits to the same fact.

**6)sometime-after -** The sometime-after state-trajectory constraint expresses that some predicate becomes true, at some point after a separate predicate becomes true.

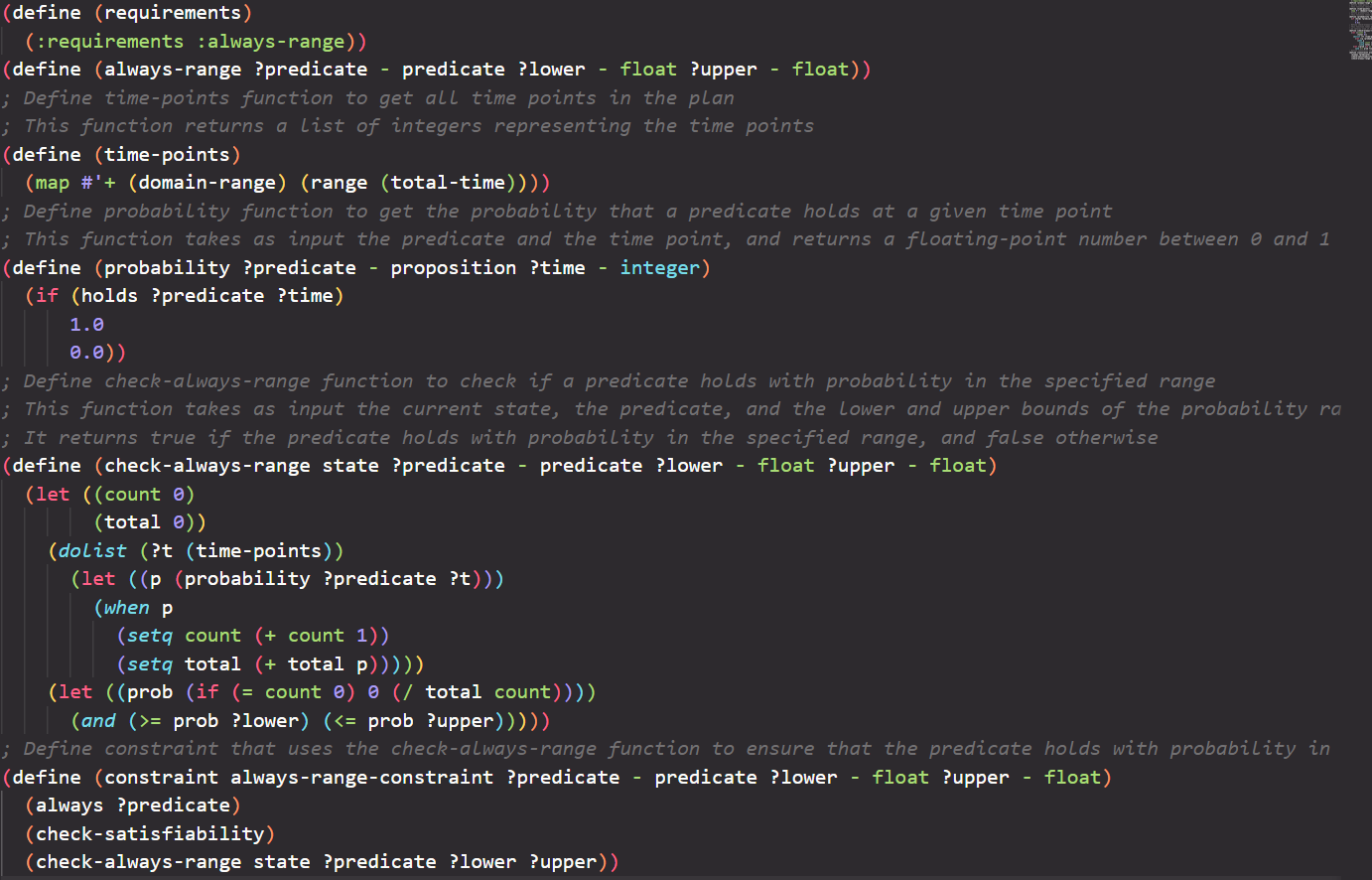
**7)sometime-before -** The sometime-before state-trajectory constraint expresses that some predicate should become true, before a separate predicate becomes true.

**8)sometime-within**

# TEMPORAL EXAMPLE



# ADDING FUNCTIONALITY IN PDDL



I wrote a LISP code for implementing a new functionality in PDDL which extends the always constraint in PDDL with adding probability range to it specifying that the probability that a particular predicate satisfies throughout the plan will lie in the provided probability range. This functionality is implemented as a new PDDL requirement called always-range.

The detail on the code is written using comments .

This functionality can be used in PDDL by adding :always-range in requirements section

# RESOURCES

IPC website: https://www.icaps-conference.org/competitions/

CP4TP temporal planner: https://planning.wiki/ref/planners/cpt, https://planning.wiki/ref/planners/cp4tp A to Z of

Planners: https://planning.wiki/ref/planners/atoz PDDL: http://www.cs.yale.edu/homes/dvm/papers/pddl-ipc5.pdf, https://arxiv.org/abs/1706.08317

PDDL Domain: https://planning.wiki/ref/pddl3/domain

IPC HOMEPAGE: https://ipc06.icaps-conference.org/deterministic/