**Batch: A-3 Roll No.: 16010122104**

**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/**](https://planning.wiki/)**, last retrieved on Feb 27,2025**
2. [**https://editor.planning.domains/**](https://editor.planning.domains/)**, last retrieved on Feb 27,2025**
3. [**https://www.youtube.com/watch?v=EeQcCs9SnhU**](https://www.youtube.com/watch?v=EeQcCs9SnhU)**, last retrieved on Feb 27,2025**
4. [**https://www.youtube.com/watch?v=FS95UjrICy0**](https://www.youtube.com/watch?v=FS95UjrICy0)**, last retrieved on Feb 27,2025**
5. [**https://nms.kcl.ac.uk/planning/software/optic.html**](https://nms.kcl.ac.uk/planning/software/optic.html)**, last retrieved on Feb 27,2025**
6. [**https://github.com/yarox/pddl-examples**](https://github.com/yarox/pddl-examples)**, last retrieved on Feb 27,2025**
7. [**https://planning.wiki/\_citedpapers/pddl3bnf.pdf**](https://planning.wiki/_citedpapers/pddl3bnf.pdf) **, last retrieved on Feb 27,2025**
8. [**https://www.youtube.com/watch?v=XW0z8Oik6G8**](https://www.youtube.com/watch?v=XW0z8Oik6G8)
9. **https://github.com/potassco/pddl-instances, last retrieved on Feb 27,2025**
10. **“Artificial Intelligence: a Modern Approach” by Russell and Norving, Pearson education Publications**
11. **“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 Library Book Management System involves tracking books, managing loans, organizing shelves, and handling patron interactions. This problem requires coordinating multiple actions such as checking books in/out, shelving books, cataloging new acquisitions, and managing overdue items.

**STRIPS/ADL Script for solving problem:**

STRIPS/ADL Script for Library Management System

Based on the search results, I can create a PDDL script for a library management system using both STRIPS and ADL features. The library management system will handle operations like adding books, checking out books, returning books, and managing patrons.

STRIPS Domain Script

text

(define (domain library-management-strips)

(:requirements :strips :typing)

(:types

book patron shelf location)

(:predicates

(book-at ?b - book ?l - location)

(on-shelf ?b - book ?s - shelf)

(checked-out ?b - book ?p - patron)

(available ?b - book)

(patron-registered ?p - patron)

(shelf-has-space ?s - shelf)

)

(:action check-out-book

:parameters (?b - book ?p - patron ?s - shelf)

:precondition (and (patron-registered ?p) (on-shelf ?b ?s) (available ?b))

:effect (and (checked-out ?b ?p) (not (on-shelf ?b ?s)) (not (available ?b)))

)

(:action return-book

:parameters (?b - book ?p - patron ?l - location)

:precondition (checked-out ?b ?p)

:effect (and (book-at ?b ?l) (not (checked-out ?b ?p)))

)

(:action shelve-book

:parameters (?b - book ?s - shelf ?l - location)

:precondition (and (book-at ?b ?l) (shelf-has-space ?s))

:effect (and (on-shelf ?b ?s) (available ?b) (not (book-at ?b ?l)))

)

(:action register-patron

:parameters (?p - patron)

:precondition (not (patron-registered ?p))

:effect (patron-registered ?p)

)

)

ADL Domain Script

text

(define (domain library-management-adl)

(:requirements :adl :typing)

(:types

book patron shelf location category - object

fiction non-fiction reference - category)

(:predicates

(book-at ?b - book ?l - location)

(on-shelf ?b - book ?s - shelf)

(checked-out ?b - book ?p - patron)

(available ?b - book)

(overdue ?b - book)

(belongs-to-category ?b - book ?c - category)

(shelf-for-category ?s - shelf ?c - category)

(patron-registered ?p - patron)

(reserved ?b - book ?p - patron)

)

(:action check-out-book

:parameters (?b - book ?p - patron ?s - shelf)

:precondition (and (patron-registered ?p)

(on-shelf ?b ?s)

(available ?b)

(or (not (exists (?p2 - patron) (reserved ?b ?p2)))

(reserved ?b ?p)))

:effect (and (checked-out ?b ?p)

(not (on-shelf ?b ?s))

(not (available ?b))

(when (reserved ?b ?p) (not (reserved ?b ?p))))

)

(:action return-book

:parameters (?b - book ?p - patron ?l - location)

:precondition (checked-out ?b ?p)

:effect (and (book-at ?b ?l)

(not (checked-out ?b ?p))

(not (overdue ?b)))

)

(:action shelve-book

:parameters (?b - book ?s - shelf ?l - location)

:precondition (and (book-at ?b ?l)

(exists (?c - category)

(and (belongs-to-category ?b ?c)

(shelf-for-category ?s ?c))))

:effect (and (on-shelf ?b ?s)

(available ?b)

(not (book-at ?b ?l)))

)

(:action reserve-book

:parameters (?b - book ?p - patron)

:precondition (and (patron-registered ?p)

(not (reserved ?b ?p))

(not (checked-out ?b ?p)))

:effect (reserved ?b ?p)

)

(:action mark-book-overdue

:parameters (?b - book ?p - patron)

:precondition (and (checked-out ?b ?p) (not (overdue ?b)))

:effect (overdue ?b)

)

)

Problem Script

text

(define (problem library-operations)

(:domain library-management-adl)

(:objects

book1 book2 book3 - book

patron1 patron2 - patron

fiction-shelf non-fiction-shelf reference-shelf - shelf

returns-desk circulation-desk - location

fiction-cat non-fiction-cat reference-cat - category

)

(:init

(book-at book1 returns-desk)

(on-shelf book2 fiction-shelf)

(on-shelf book3 reference-shelf)

(belongs-to-category book1 fiction-cat)

(belongs-to-category book2 fiction-cat)

(belongs-to-category book3 reference-cat)

(shelf-for-category fiction-shelf fiction-cat)

(shelf-for-category non-fiction-shelf non-fiction-cat)

(shelf-for-category reference-shelf reference-cat)

(available book2)

(available book3)

(patron-registered patron1)

)

(:goal

(and

(on-shelf book1 fiction-shelf)

(checked-out book2 patron1)

(patron-registered patron2)

)

)

)

Key Differences Between STRIPS and ADL Implementations

1. Expressivity: The ADL version includes conditional effects (when clause) and quantified formulas (exists), which aren't available in STRIPS.
2. Preconditions: ADL allows disjunctive preconditions (or) and negative preconditions, while STRIPS only supports conjunctive positive preconditions.
3. Reservation System: The ADL version implements a book reservation system with conditional logic that isn't possible in basic STRIPS.
4. Category Management: The ADL version includes a more sophisticated category system for books and shelves, leveraging the greater expressivity of ADL.

**PDDL Script for solving problem:**

PDDL Domain Script

(define (domain library-management)

(:requirements :strips :typing)

(:types

book patron shelf location category - object

fiction non-fiction reference - category)

(:predicates

(book-at ?b - book ?l - location)

(on-shelf ?b - book ?s - shelf)

(checked-out ?b - book ?p - patron)

(available ?b - book)

(overdue ?b - book)

(reserved ?b - book ?p - patron)

(belongs-to-category ?b - book ?c - category)

(shelf-for-category ?s - shelf ?c - category)

(shelf-has-space ?s - shelf)

(cataloged ?b - book)

(patron-registered ?p - patron)

)

(:action check-out-book

:parameters (?b - book ?p - patron ?s - shelf)

:precondition (and (patron-registered ?p) (on-shelf ?b ?s) (available ?b))

:effect (and (checked-out ?b ?p) (not (on-shelf ?b ?s)) (not (available ?b)))

)

(:action return-book

:parameters (?b - book ?p - patron ?l - location)

:precondition (checked-out ?b ?p)

:effect (and (book-at ?b ?l) (not (checked-out ?b ?p))

(not (overdue ?b)))

)

(:action shelve-book

:parameters (?b - book ?s - shelf ?l - location ?c - category)

:precondition (and (book-at ?b ?l) (belongs-to-category ?b ?c)

(shelf-for-category ?s ?c) (shelf-has-space ?s))

:effect (and (on-shelf ?b ?s) (available ?b) (not (book-at ?b ?l)))

)

(:action catalog-new-book

:parameters (?b - book ?c - category ?l - location)

:precondition (and (book-at ?b ?l) (not (cataloged ?b)))

:effect (and (cataloged ?b) (belongs-to-category ?b ?c))

)

(:action mark-book-overdue

:parameters (?b - book ?p - patron)

:precondition (and (checked-out ?b ?p) (not (overdue ?b)))

:effect (overdue ?b)

)

(:action reserve-book

:parameters (?b - book ?p - patron)

:precondition (and (patron-registered ?p) (not (reserved ?b ?p)))

:effect (reserved ?b ?p)

)

(:action cancel-reservation

:parameters (?b - book ?p - patron)

:precondition (reserved ?b ?p)

:effect (not (reserved ?b ?p))

)

(:action register-patron

:parameters (?p - patron)

:precondition (not (patron-registered ?p))

:effect (patron-registered ?p)

)

)

PDDL Problem Script

(define (problem library-daily-operations)

(:domain library-management)

(:objects

book1 book2 book3 book4 book5 - book

patron1 patron2 patron3 - patron

fiction-shelf non-fiction-shelf reference-shelf - shelf

returns-desk processing-area circulation-desk - location

fiction-cat non-fiction-cat reference-cat - category

)

(:init

; Books and their locations

(book-at book1 returns-desk)

(book-at book2 processing-area)

(on-shelf book3 fiction-shelf)

(on-shelf book4 reference-shelf)

(checked-out book5 patron1)

; Categories

(belongs-to-category book1 fiction-cat)

(belongs-to-category book3 fiction-cat)

(belongs-to-category book4 reference-cat)

(belongs-to-category book5 non-fiction-cat)

; Shelf configurations

(shelf-for-category fiction-shelf fiction-cat)

(shelf-for-category non-fiction-shelf non-fiction-cat)

(shelf-for-category reference-shelf reference-cat)

(shelf-has-space fiction-shelf)

(shelf-has-space non-fiction-shelf)

; Book statuses

(available book3)

(available book4)

(cataloged book1)

(cataloged book3)

(cataloged book4)

(cataloged book5)

; Patron statuses

(patron-registered patron1)

(patron-registered patron2)

; Reservations

(reserved book3 patron2)

)

(:goal

(and

(on-shelf book1 fiction-shelf)

(cataloged book2)

(belongs-to-category book2 non-fiction-cat)

(on-shelf book2 non-fiction-shelf)

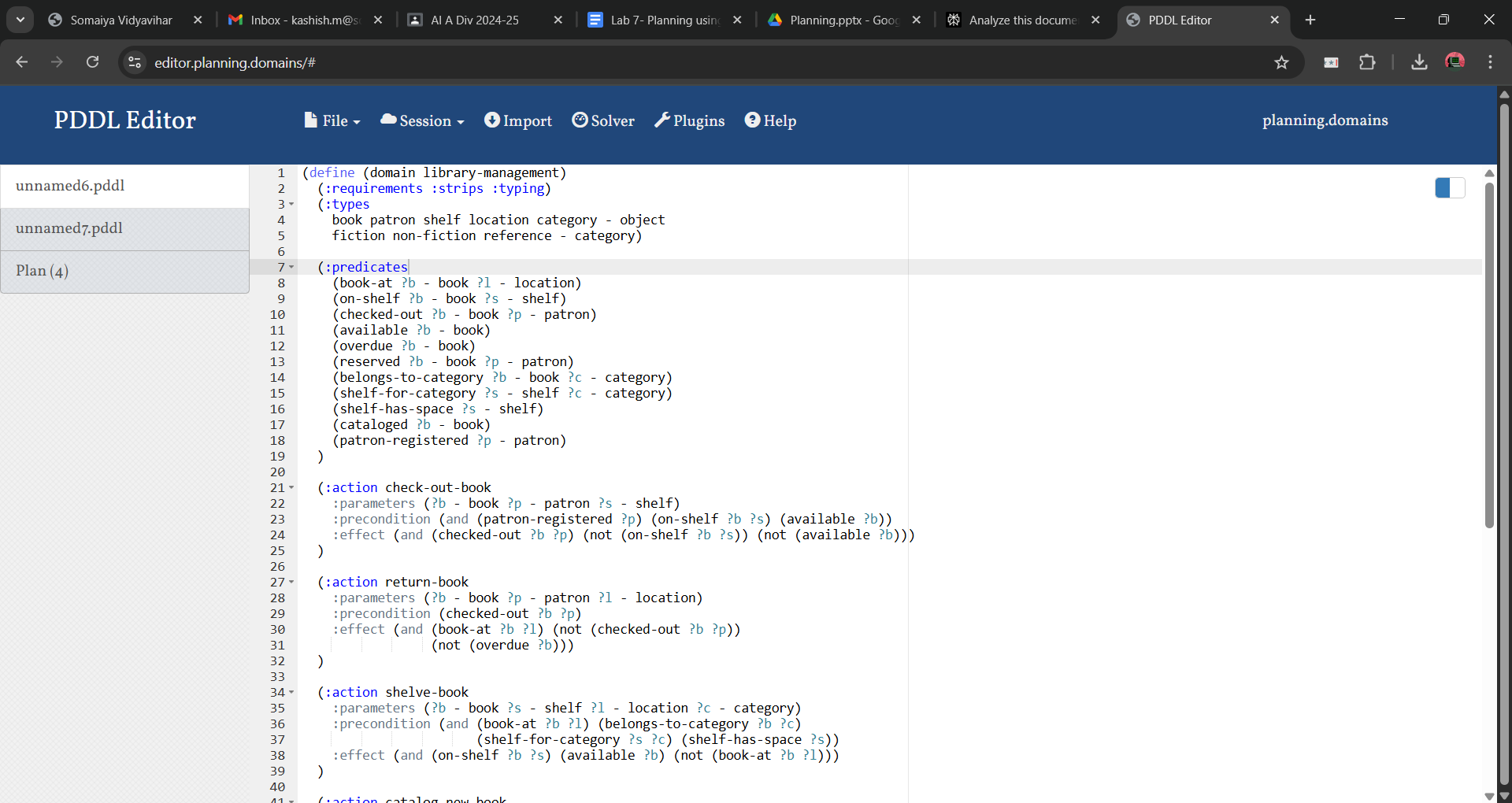
(checked-out book3 patron2)

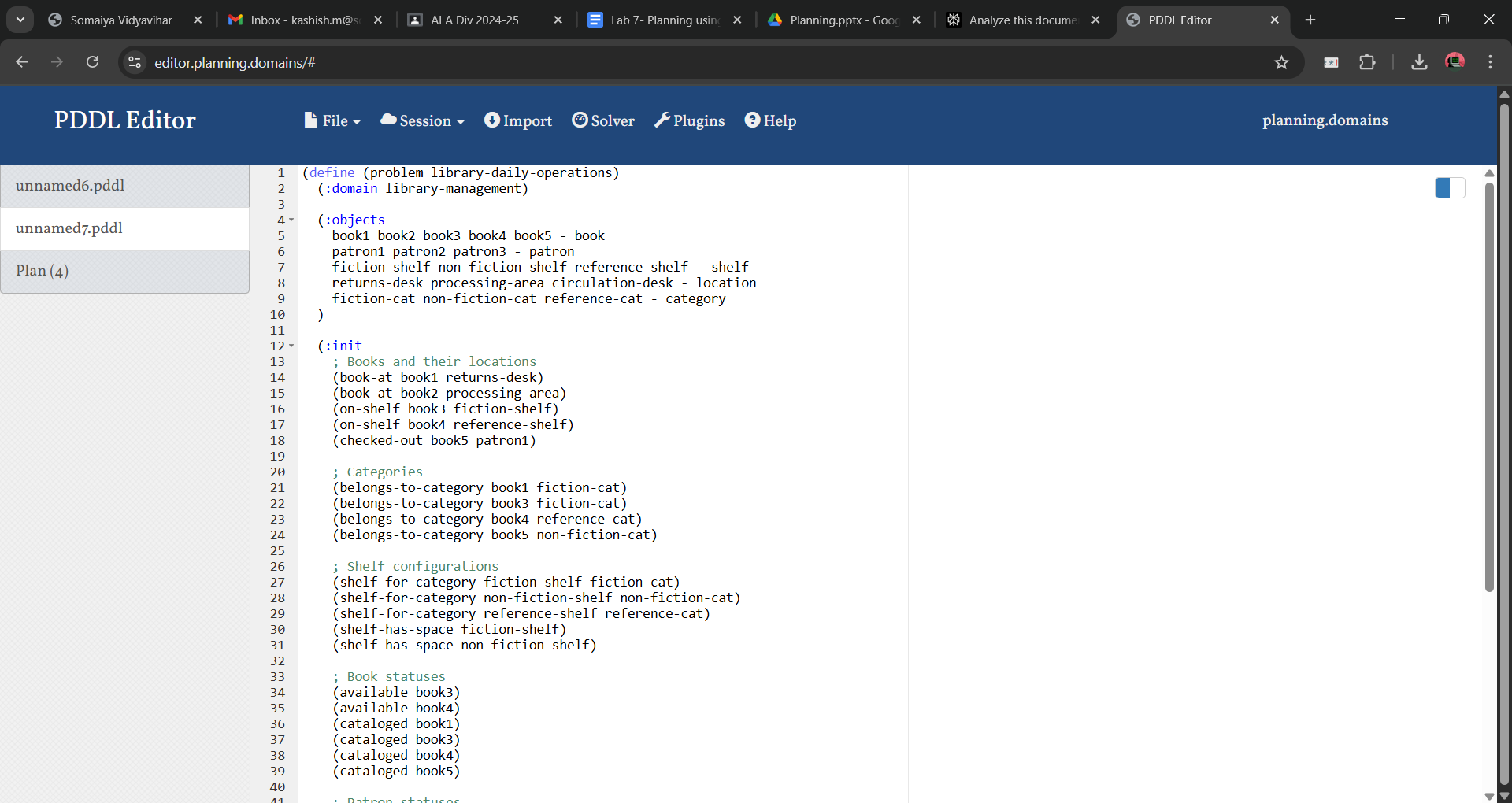
(patron-registered patron3)

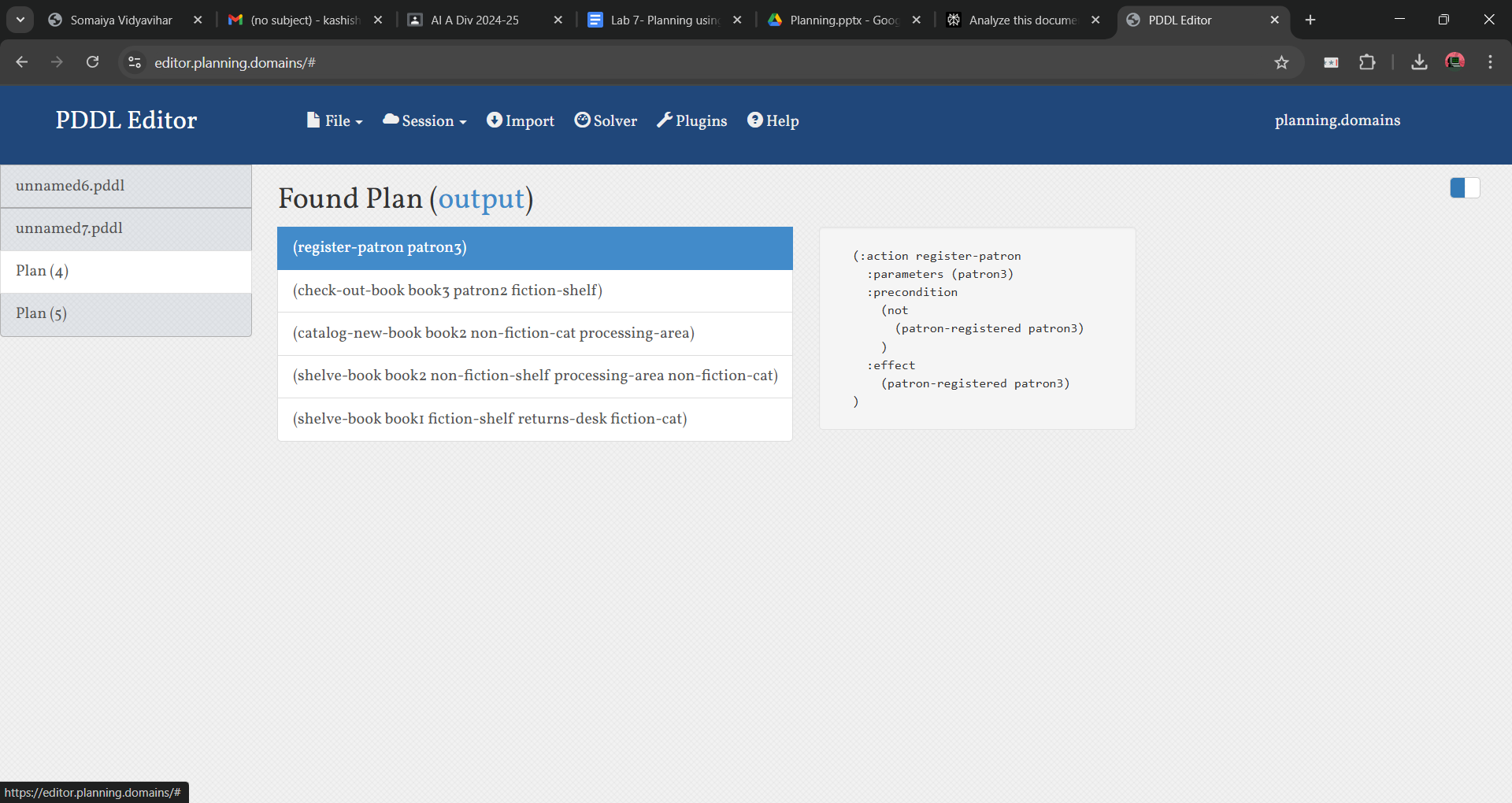
)

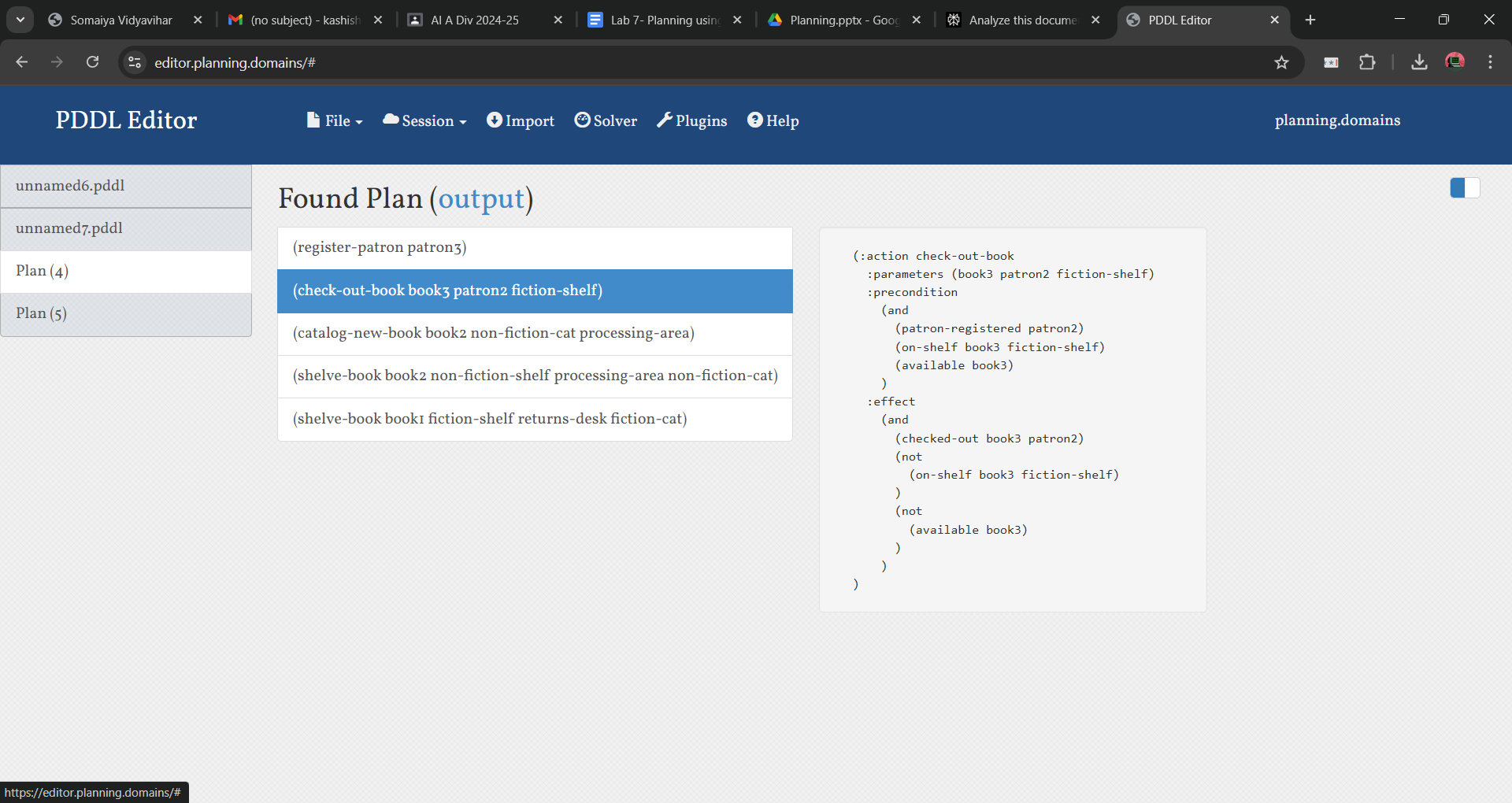
)

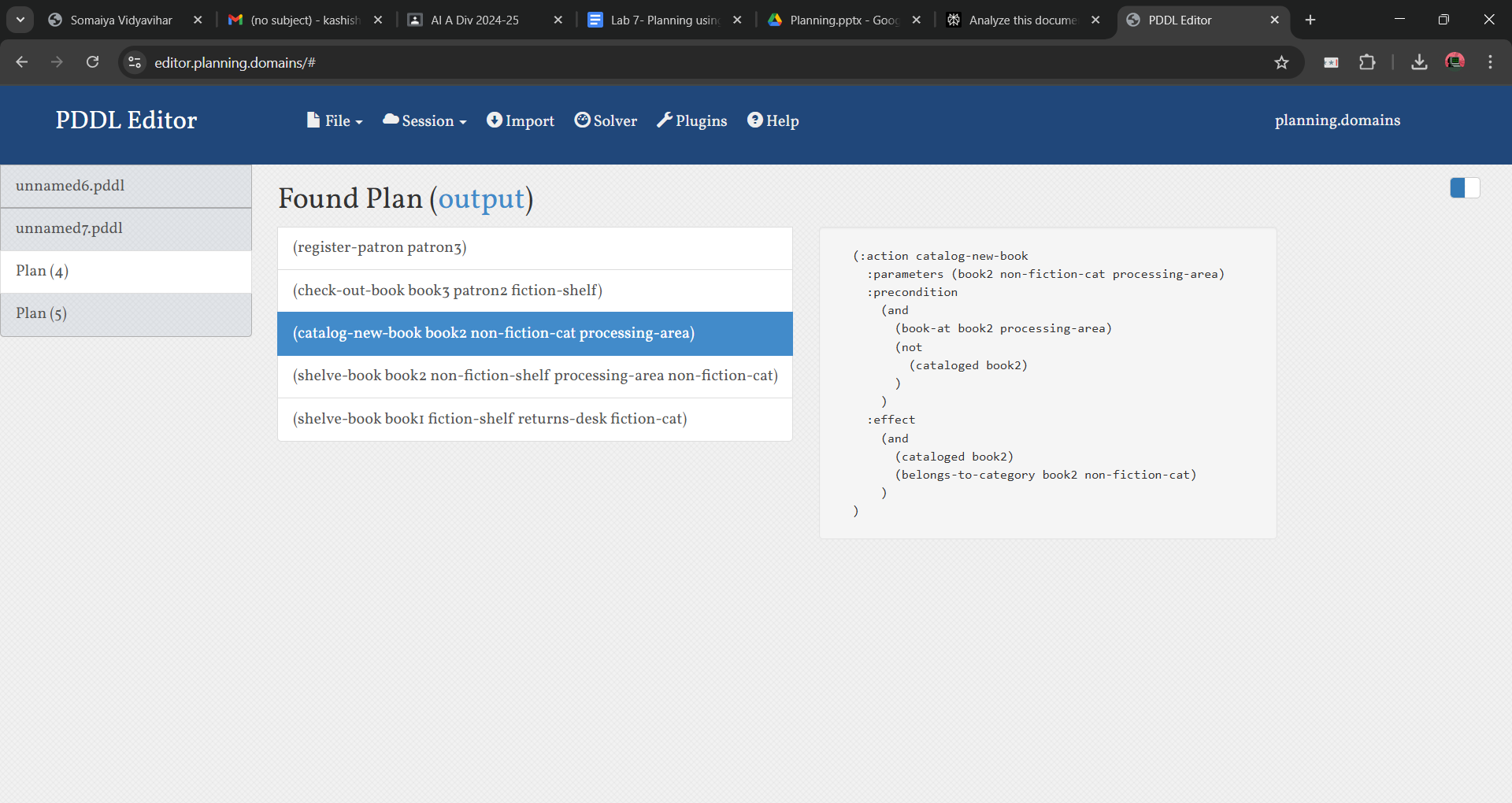
)

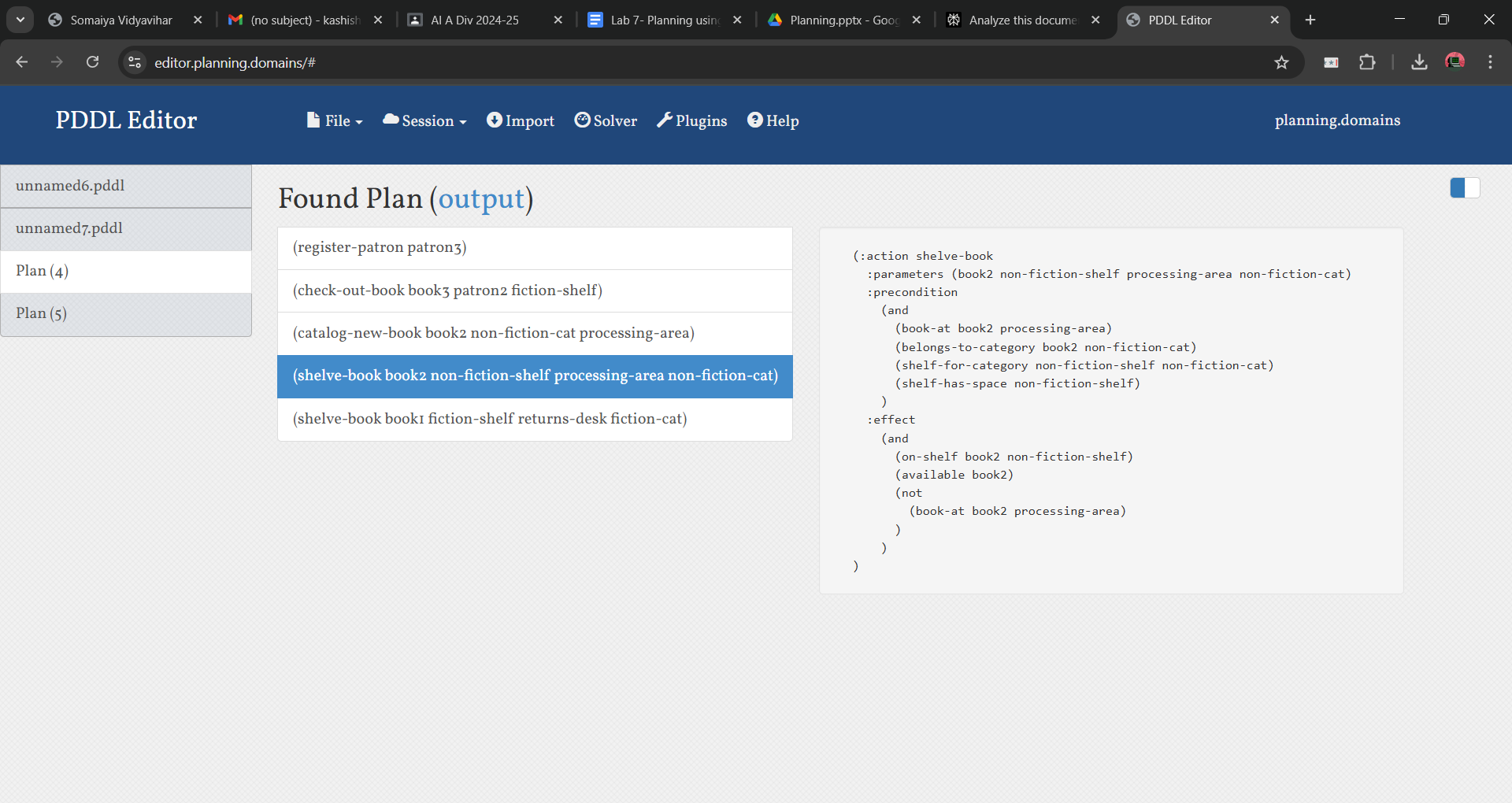


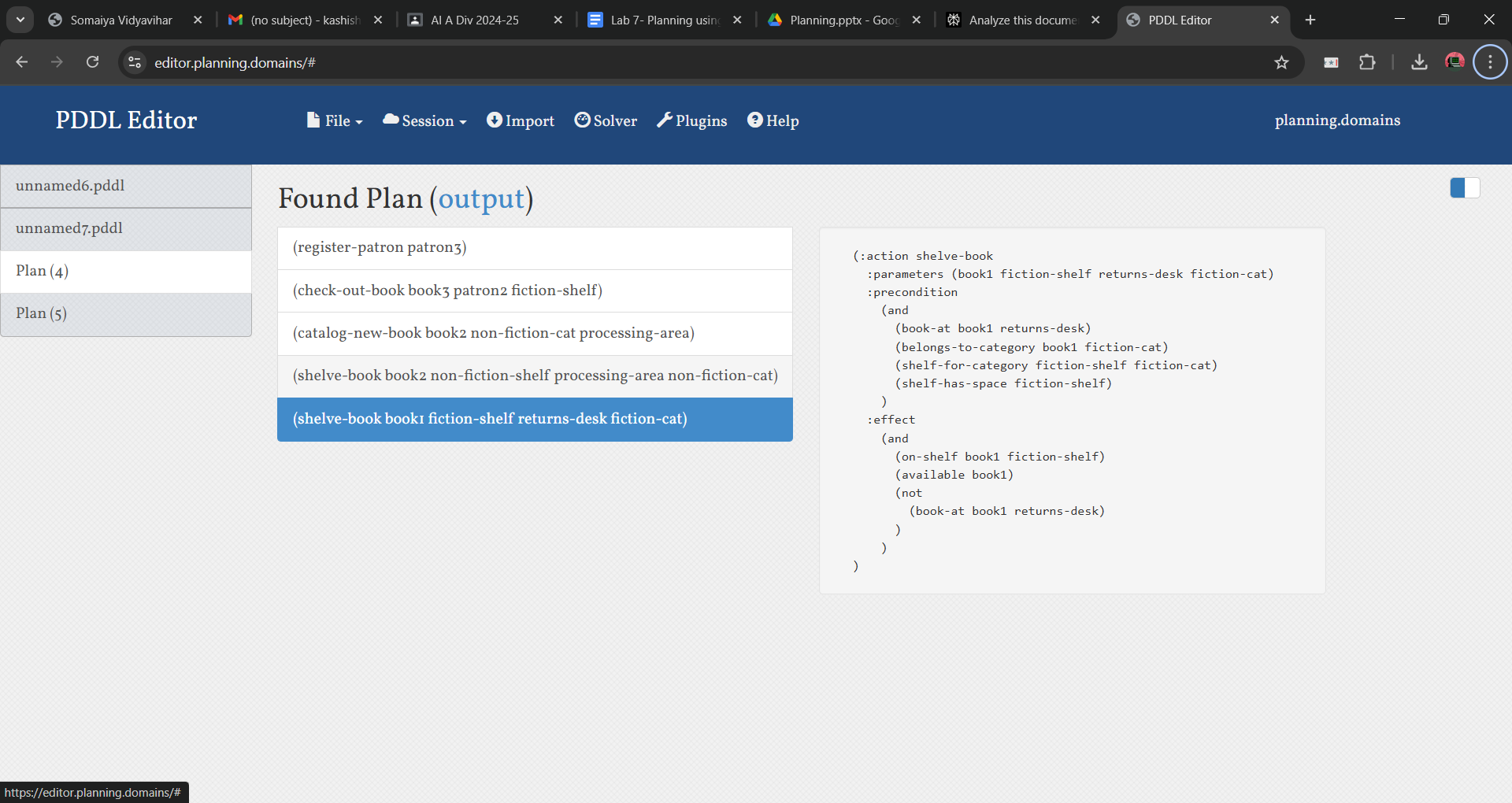












**Explanation of PDDL Model:**

PDDL (Planning Domain Definition Language) is a standardized language for describing planning problems in artificial intelligence. It's used to define domains and problems for automated planning systems. Here's a detailed explanation of the key components in a PDDL model:

**Domain Definition**

The domain definition describes the general structure of the problem space.

text

(define (domain library-management)

(:requirements :strips :typing)

(:types

book patron shelf location category - object

fiction non-fiction reference - category)

* **:requirements**: Specifies the PDDL features used in the domain.
* **:types**: Defines the types of objects in the domain, allowing for hierarchical typing.

**Predicates**

Predicates define the possible states or relations between objects.

text

(:predicates

(book-at ?b - book ?l - location)

(on-shelf ?b - book ?s - shelf)

(checked-out ?b - book ?p - patron)

(available ?b - book)

...

)

Each predicate represents a fact that can be true or false about objects in the domain.

**Actions**

Actions define how the state of the world can change.

text

(:action check-out-book

:parameters (?b - book ?p - patron ?s - shelf)

:precondition (and (patron-registered ?p) (on-shelf ?b ?s) (available ?b))

:effect (and (checked-out ?b ?p) (not (on-shelf ?b ?s)) (not (available ?b)))

)

* **:parameters**: Variables used in the action.
* **:precondition**: Conditions that must be true for the action to be applicable.
* **:effect**: Changes to the world state when the action is executed.

**Problem Definition**

The problem definition describes a specific instance within the domain.

text

(define (problem library-daily-operations)

(:domain library-management)

(:objects

book1 book2 book3 book4 book5 - book

patron1 patron2 patron3 - patron

...

)

* **:objects**: Declares specific objects in the problem instance.

**Initial State**

Defines the starting conditions of the problem.

text

(:init

(book-at book1 returns-desk)

(on-shelf book3 fiction-shelf)

(checked-out book5 patron1)

...

)

**Goal State**

Specifies the conditions to be achieved.

text

(:goal

(and

(on-shelf book1 fiction-shelf)

(cataloged book2)

(checked-out book3 patron2)

...

)

)

**Key Aspects of PDDL Modeling**

1. **Abstraction**: PDDL allows for modeling complex real-world scenarios in a simplified, abstract form.
2. **Flexibility**: The domain-problem separation allows for solving multiple problems within the same domain.
3. **Expressiveness**: PDDL can represent a wide range of planning problems, from simple to highly complex.
4. **Modularity**: Actions and predicates can be reused and combined in various ways.
5. **Scalability**: PDDL models can be easily expanded by adding new types, predicates, or actions.

**Post Lab Descriptive Questions:**

**1. How does ADL (Action Description Language) extend STRIPS?**

**STRIPS (Stanford Research Institute Problem Solver) is a simple planning language that uses propositional logic to represent states and actions. However, it has limitations. ADL extends STRIPS by adding several features, making it more expressive and capable of handling more complex planning problems. Here's a breakdown:**

* **Disjunctive Preconditions and Goals: STRIPS only allows conjunctions (ANDs) in preconditions and goals. ADL allows disjunctions (ORs), enabling plans to handle situations where multiple conditions can satisfy a requirement.**
* **Negated Preconditions and Goals: STRIPS only allows positive literals. ADL allows negated literals, which are essential for representing situations where something is *not* true.**
* **Quantifiers: ADL introduces universal (∀) and existential (∃) quantifiers, allowing the representation of actions and goals that apply to sets of objects.**
* **Types and Variables: ADL supports object types and variables, making it easier to represent and reason about objects and their relationships.**
* **Conditional Effects: ADL allows conditional effects, where the outcome of an action depends on the state of the world. This is represented as "when... then..." clauses.**
* **Functions: ADL supports functions, which allow for the representation of numerical or symbolic values that can change as a result of actions. This allows for planning in domains with numerical resources or other dynamic values.**

**In essence, ADL significantly enhances the expressiveness of STRIPS, enabling the representation of a wider range of planning problems.**

**2. Define Partial Order Planning (POP) and Total Order Planning (TOP). How do they differ?**

* **Total Order Planning (TOP):**
  + **TOP creates a plan where all actions are strictly ordered in a linear sequence.**
  + **It commits to a specific sequence of actions from the beginning.**
  + **It is often simpler to implement but can be inefficient if the order of some actions is not critical.**
  + **Example: STRIPS planning typically produces total order plans.**
* **Partial Order Planning (POP):**
  + **POP creates a plan where only the necessary ordering constraints between actions are specified.**
  + **It allows for flexibility in the order of actions, as long as the constraints are satisfied.**
  + **It can be more efficient because it avoids committing to unnecessary ordering constraints.**
  + **It can handle situations where the optimal order of actions is not known in advance.**
  + **POP handles threats, which is a situation where an action will negate a precondition of another action that is supposed to be true.**

**Differences:**

* **Ordering: TOP imposes a complete linear ordering, while POP allows partial ordering.**
* **Flexibility: POP is more flexible, allowing for exploration of a wider range of plan possibilities. TOP is more rigid.**
* **Efficiency: POP can be more efficient for problems where the order of some actions is not critical. TOP can be less efficient because it may explore unnecessary orderings.**
* **Threat Handling: POP is designed to handle threats, TOP does not handle threats within its basic form.**

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

**Whether POP is beneficial depends heavily on the specifics of the planning problem. Here's a general guideline:**

* **When POP is beneficial:**
  + **Problems with many independent actions where the order of some actions doesn't matter.**
  + **Problems where the optimal order of actions is not known in advance.**
  + **Problems that include threats that need to be resolved.**
  + **Problems with non-linear dependencies between actions.**
* **When POP might be less beneficial:**
  + **Simple problems with a fixed, linear sequence of actions.**
  + **Problems where efficiency is not a critical concern and a simple total order plan is sufficient.**

**If a problem has many actions that can be performed in various orders, and if there are potential threats that need to be resolved, then POP would likely be beneficial. If the problem is simple and linear, total order planning is sufficient.**

**4. Correlate the knowledge engineering steps with PDDL scripts.**

**PDDL (Planning Domain Definition Language) is a standard language for describing planning problems. The knowledge engineering steps involved in creating a PDDL script are:**

1. **Identify the goal:**
   * **In PDDL, this is represented in the :goal section of the problem file.**
   * **Example: (:goal (and (at robot location2) (has-item robot item1)))**
2. **Identify the initial state:**
   * **In PDDL, this is represented in the :init section of the problem file.**
   * **Example: (:init (at robot location1) (carrying robot nothing) (at item1 location3))**
3. **Identify the actions:**
   * **In PDDL, actions are defined in the domain file using the :action keyword.**
   * **Each action has a :parameters, :precondition, and :effect section.**
   * **Example:**

**Code snippet**

**(:action move**

**:parameters (?r - robot ?from - location ?to - location)**

**:precondition (and (at ?r ?from) (adjacent ?from ?to))**

**:effect (and (at ?r ?to) (not (at ?r ?from)))**

**)**

1. **Identify the objects and their types:**
   * **In PDDL, objects and their types are defined in the problem file using the :objects keyword.**
   * **Types are defined in the domain file using the :types keyword.**
   * **Example:**

**Code snippet**

**(:domain my-domain)**

**(:types robot location item)**

**(:objects robot1 - robot location1 location2 location3 - location item1 - item)**

1. **Identify the predicates and functions:**
   * **In PDDL, predicates and functions are defined in the domain file using the :predicates and :functions keywords.**
   * **Predicates represent relationships or properties, while functions represent numerical or symbolic values.**
   * **Example:**

**Code snippet**

**(:predicates (at ?obj - object ?loc - location) (carrying ?r - robot ?item - object) (adjacent ?loc1 - location ?loc2 - location))**

**In summary, the knowledge engineering process is directly translated into the structural elements of PDDL scripts, allowing for the formal representation of planning problems.**