Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfilment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

M. Yaswanth Sai- AP22110011084

K. Ravi Prakash Reddy- AP22110011091

S. Jayanth Kumar-AP22110011109

L. Sahithya -AP22110011089

Under the Guidance of

Dr. Ajay Dilip Kumar Marapatla

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 503

April, 2025

ACKNOWLEDGEMENT

We would like to express our deep sense of gratitude and honour to our beloved faculty supervisor, **Dr Ajay Dilip Kumar Marapatla** sir for his constant encouragement and priceless intellectual motivation throughout my project period under his guidance and supervision. Their academic excellence, critical reviews and constructive comments improved my grasp of the subject and steered to the fruitful completion of the work. His patience, guidance and encouragement made this project possible.

We are truly grateful for his patience, kind support, and motivation at every step. It was a great learning experience and an honour to work under his guidance.

CONTENTS

ABSTRACT

Efficient course scheduling is a critical administrative task in educational institutions, requiring the optimal allocation of time slots for theory and lab sessions across multiple courses, while adhering to institutional constraints and faculty availability. This project presents an **AI-driven Course Timetable Scheduling System** built using **Prolog**, a logic programming language particularly suited for solving combinatorial and constraint-based problems. The system leverages the principles of **Artificial Intelligence**, especially **Constraint Satisfaction Problems (CSP)**, to intelligently assign time slots for courses based on defined rules and constraints such as lecture durations, lab requirements, and special cases like fixed slots for certain subjects.

We have considered our own courses for this semester i.e.3Rd year 2nd Semester for this project and set Prolog rules and constraints based on them.

The proposed system includes predefined time slots (excluding lunch breaks and blocked hours like Wednesday Afternoons) and categorizes each course by its lecture and lab hour requirements. Special considerations are given to certain subjects such as *CSE437*, which must occur during a fixed evening time, and *SEC137*, which may be scheduled either as three consecutive theory hours or split into two sessions. The system intelligently manages these constraints using Prolog's built-in backtracking and logical inference capabilities. The solution generates both theory and lab session allocations, ensuring no overlapping or violation of time slot availability.

From an **AI perspective**, the project represents a classic example of a **single-agent system** operating in a **fully observable**, **deterministic**, and **static environment**. The agent has full knowledge of the time slots and course requirements, and there is no uncertainty or dynamic change once the scheduling begins. The Prolog engine acts as the **problem-solving agent**, searching through possible configurations and eliminating those that violate the given constraints.

This project not only demonstrates the practical utility of AI in academic operations but also highlights the elegance of declarative programming in solving real-world scheduling problems.

INTRODUCTION

Scheduling a university timetable is a complex task that involves assigning courses to time slots while adhering to multiple constraints, such as avoiding conflicts, respecting time preferences, and accommodating breaks. Traditional manual scheduling is time-consuming and error-prone, especially when dealing with specialized constraints like consecutive class hours or restricted time windows. This project develops an automated timetable scheduler for seven courses at SRM University AP, using GNU Prolog to model and solve the problem as a CSP.

APPROACH

The timetable scheduling problem is formulated as a CSP, where:

- **Variables:** Represent theory classes (21 one-hour slots) and lab sessions (4 two-hour slots) for courses CSE 306, CSE 307, CSE 437, CSE 455, CSE 456, MAT 273, and SEC 137.
- **Domains**: 66 thirty-minute periods (9:00 AM–5:30 PM, Monday–Friday, excluding 1:00–2:00 PM lunch and Wednesday 1:00–5:30 PM), grouped into 33 one-hour slots for theory and contiguous pairs for labs.

CONSTRAINTS

- **No overlap**: Each time slot is assigned to at most one class.
- **CSE 437**: Two one and half hour theory classes must occur between 4:00–5:30 PM on Any of the two days from Monday, Tuesday, Thursday, Friday.
- **SEC 137**: Three one-hour theory classes must be consecutive (e.g., Monday 10:00–13:00).
- **Lunch break**: No classes from 1:00–2:00 PM daily.
- **Wednesday afternoon**: Free from 1:00–5:30 PM.(For Clubs)
- **Lab Continuity**: Labs Must take a continuous 2hr Class (i.e. 4*30 mins).

SOLUTION STRATEGY

GNU Prolog's declarative programming allows us to define the problem using predicates (e.g., slot/2, hour_slot/2, lab_slot/4, course/3) and constraints (e.g., select/3 for slot assignment, append/3 for combining assignments). The timetable/1 predicate assigns slots to classes using backtracking, ensuring all constraints are satisfied. Debugging predicates (e.g., write/1 for failure messages) help identify issues, and debugging them such as the challenge faced append/2 error while programming, which was resolved by using GNU Prolog's append/3.

OBJECTIVE

- The objective of this project is to develop an automated timetable scheduler using GNU Prolog that:
- Generates a valid timetable for the seven courses, assigning 21 theory classes and 4 lab sessions to 66 thirty-minute periods.
- Satisfies all constraints (no overlaps, CSE 437's time restrictions, SEC 137's consecutive hours, lunch break, Wednesday afternoon free).
- Produces a console-based output listing course assignments (e.g., course, type, time slot).

Demonstrates the application of logic programming and CSP in solving real-world scheduling problems, aligning with SRM University AP Timetable scheduling, Through simple AI.

TECHNOLOGY STACK

- **GNU Prolog (Version 1.5.0):** A free, open-source Prolog compiler optimized for constraint solving and logic programming. It supports predicates like findall/3, select/3, between/3, and append/3, which are used to model the CSP. GNU Prolog's backtracking and finite domain constraint solver efficiently handle the 24–25 variables and 66 periods .No frontend or additional technologies (e.g., MERN stack) are used, as the focus is on backend logic and console-based output.
- **Notepad++:** Text Editor to write the Prolog logic as a Perl File (i.e. **.pl** file).
- **Development Environment**: Windows 10, GNU Prolog console for testing and debugging, with the program stored as **timetable.pl**.

This approach leverages GNU Prolog's strengths in declarative programming, making the solution concise, maintainable.

DATA

Course	Theory Hours	Lab Hours
cse306	3	2
cse307	3	2
cse437	3	0
cse455	3	2
cse456	3	2
mat273	3	0
sec137	3	0

WHAT TYPE OF AGENT AND ENVIRONMENT

Agent:

The timetable scheduler is a **Knowledge-Based Agent** which is also called as *Logical Agent*.

The agent we created using Prolog operates by **storing facts and rules** in a **knowledge base** and **reasoning logically** to derive conclusions or make decisions.
It does not act randomly or learn from experience, but instead infers new information purely based on predefined logical relationships.
This matches exactly with the characteristics of a knowledge-based agent, which relies on logical inference to determine its actions.

Environment:

Fully Observable: The agent has complete knowledge of the available slots (slot/2, hour_slot/2), courses (course/3), and constraints at the start.

Deterministic: Given a set of assignments, the outcome (valid or invalid timetable) is predictable, with no randomness.

Static: The environment (slots, courses, constraints) does not change during execution.

Discrete: Time slots are discrete (30-minute periods), and assignments are finite.

Single-Agent: No other agents interact; the scheduler operates independently.

AGENT ARCHITECTURE

1. Perception (Sensors):

- **Slot Availability**: This module will keep track of all available time slots (slot/2), hour_slot/2 (which defines the length of time blocks) to perceive the current state of the available slots.
- **Course Information**: It will keep track of all the courses (course/3), their required hours for theory and lab, and any special constraints (e.g., CSE 437 and SEC 137).
- **Course Constraints:** This includes constraints like lab hours, theory hours, special constraints for courses like CSE 437, ensuring that courses fit within the available slots.

2. Knowledge Base:

- **Facts**: Contains all predefined facts, such as available slots (slot/2), hours required (hour_slot/2), and course details (course/3).
- **Rules**: Defines the logical relationships and constraints, including:
    - How theory and lab hours should be allocated.
    - How special courses like CSE 437 and SEC 137 should be scheduled.
    - Rules for slot conflict checks and validation.
    - Rules for slot allocation (theory vs. lab hours).
- **Inference Engine**: This is where the agent uses logical reasoning to make decisions. The agent will apply the rules to derive conclusions about which slots are valid for each course and when to assign them.

3. Reasoning (Deliberation):

- **Decision Making**: The agent will use a rule-based reasoning process to choose the most appropriate time slots for courses, starting with theory and then moving to lab slots.

- **Conflict Resolution**: The agent will identify any conflicts (e.g., overlapping slots) and use logical rules to resolve them.
- **Constraint Handling**: It will handle constraints such as courses requiring specific time blocks, lab vs. theory slots, and special cases.

4. Action (Effectors):

- **Slot Assignment**: Once a valid time slot is determined, the agent will assign the course to that slot. This involves updating the timetable.
- **Conflict Checking**: The agent will continuously check for conflicts between course assignments and adjust accordingly.
- **Update Knowledge Base**: After a successful assignment, the agent updates the knowledge base to reflect the new state of the available slots (i.e., the assigned slots will no longer be available for other courses).

5. Learning and Adaptation:

- While our agent is non-learning (as per the knowledge-based agent design), But it can still adapt to feedback loops during scheduling. If the agent encounters conflicts, it might adjust by just backtracking in the reasoning process or recalculating available slots.

PEAS FRAMEWORK

The PEAS (Performance Measure, Environment, Actuators, Sensors) framework describes the agent as follows:

1. Performance Measure:

The performance measure defines how the success of the agent is evaluated. For the timetable scheduler, the performance measure can include:

- **Timetable Validity**: The timetable is valid if all courses are assigned to appropriate slots (no conflicts, all constraints are satisfied).
- **Slot Utilization**: The more efficiently the available slots are used (without under-utilization), the better the performance.
- **Constraint Satisfaction**: The timetable should respect all constraints, including:
    - Theory and lab hour requirements.
    - Special rules for courses like CSE 437.
    - No overlapping slots.
- **Optimization of Free Time**: Minimizing gaps between scheduled courses.
- **Fair Distribution of Slots**: Ensuring courses are distributed evenly across available times, avoiding excessive clustering or gaps for any particular course.

2. Environment:

The environment is the context in which the agent operates, and for the timetable scheduler, it includes:

- **Course Information**:
    - Each course has a unique identifier and associated requirements (theory hours, lab hours, etc.).
- **Available Slots**:
    - Time slots (slot/2) are finite and discrete (30-minute periods).

    - Specific time blocks for theory and lab sessions (hour_slot/2).
- **Course Constraints**:
    - Some courses may have special scheduling requirements (e.g., CSE 437 requiring back-to-back lab sessions).
    - Some courses may not be scheduled on certain days.
- **State of Assignments**:
    - Information about which slots are already filled by which courses.
- **Constraints and Rules**:
    - Constraints for slot allocation (e.g., no overlapping courses, max. hours per day).

3. Actuators:

Actuators are the mechanisms through which the agent interacts with the environment. In this case, the actuators are:

- **Timetable Assignment**: The agent assigns a course to an available slot. This action updates the timetable by filling slots.
- **Slot Conflict Resolution**: The agent adjusts the slot assignments to resolve conflicts (e.g., by shifting a course to another available time).
- **Knowledge Base Update**: After each slot assignment, the knowledge base is updated to reflect the newly assigned slots (no longer available for other courses).

4. Sensors:

Sensors gather information from the environment that the agent needs to make decisions. In this case, the sensors provide the agent with:

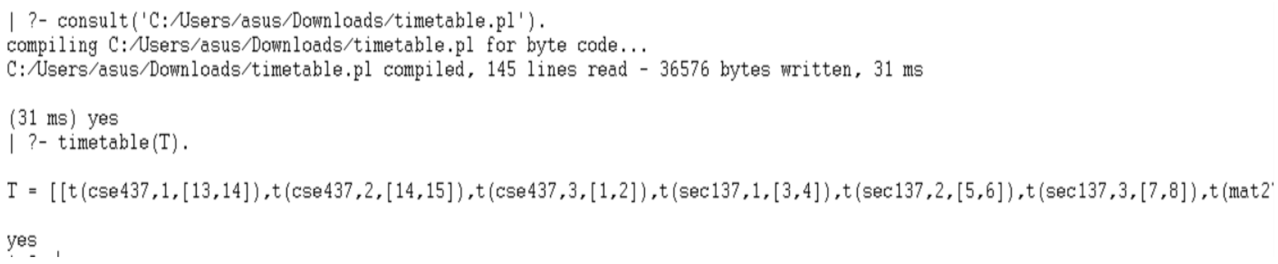
- **Available Slot Information**: The current state of available time slots (slot/2, hour_slot/2), including which slots are filled and which are free.
- **Course Information**: The properties of courses (course/3), such as the required theory and lab hours, and any special scheduling constraints.

- **Constraints State**: Whether specific constraints (e.g., CSE 437 special case) have been satisfied or violated.
- **Conflict Detection**: Information about conflicts in the current slot assignments, such as overlapping or invalid time allocations.

Summary of PEAS Framework:

PEAS Component	Details
Performance Measure	1. Timetable validity (valid assignments with no conflicts). 2. Efficient slot utilization. 3. Constraint satisfaction (including special constraints). 4. Minimization of time gaps. 5. Fair distribution of course slots.
Environment	1. Courses with unique identifiers and requirements. 2. Discrete, finite time slots. 3. Constraints and rules (e.g., no overlapping slots, theory/lab hour requirements). 4. Course-specific scheduling constraints (e.g., CSE 437 special cases).
Actuators	1. Assigning courses to time slots. 2. Resolving slot conflicts. 3. Updating the knowledge base after assignments.
Sensors	1. Available slot information (filled vs. free). 2. Course information (theory/lab hours, constraints). 3. Constraints state. 4. Conflict detection (overlapping slots, invalid assignments).

OUTPUT



```
| ?- consult('C:/Users/asus/Downloads/timetable.pl').
compiling C:/Users/asus/Downloads/timetable.pl for byte code...
C:/Users/asus/Downloads/timetable.pl compiled, 145 lines read - 36576 bytes written, 31 ms

(31 ms) yes
| ?- timetable(T).

T = [[t(cse437,1,[13,14]),t(cse437,2,[14,15]),t(cse437,3,[1,2]),t(sec137,1,[3,4]),t(sec137,2,[5,6]),t(sec137,3,[7,8]),t(mat2

yes
```

Figure 1:Output from GNU Prolog

Output:

T =

[[t(cse437,1,[13,14]),t(cse437,2,[14,15]),t(cse437,3,[1,2]),t(sec137,1,[3,4]),t(sec137,2,[5,6]),t(sec137,3,[7,8]),t(mat273,3,[54,55]),t(mat273,2,[52,53]),t(mat273,1,[50,51]),t(cse456,3,[49,50]),t(cse456,2,[47,48]),t(cse456,1,[45,46]),t(cse455,3,[43,44]),t(cse455,2,[41,42]),t(cse455,1,[39,40]),t(cse307,3,[37,38]),t(cse307,2,[35,36]),t(cse307,1,[33,34]),t(cse306,3,[31,32]),t(cse306,2,[29,30]),t(cse306,1,[28,29])],[l(cse456,[24,25,26,27]),l(cse455,[20,21,22,23]),l(cse307,[16,17,18,19]),l(cse306,[9,10,11,12])]]

FUTURE EXTENSION

1. Incorporating Additional Constraints:

To make the timetable scheduler more robust, additional constraints can be added, such as:

- **Instructor Constraints**: Consider instructor availability and preferences when assigning courses to slots.
- **Student Group Constraints**: Handle student-specific course enrolments to avoid scheduling conflicts.
- **Room or Resource Constraints**: Ensure that rooms and resources are available when assigning courses, considering room capacities and availability.

2. Handling Different Sections or Departments:

The system can be extended to manage multiple sections or departments simultaneously:

- **Department-Specific Timetables**: Each department (e.g., Computer Science, Mechanical Engineering) can have its own set of rules and constraints, allowing for tailored scheduling.
- **Multiple Course Sections**: The system can handle multiple sections for large courses, ensuring that sections do not overlap with each other or other courses.

3. Integration with a Graphical User Interface (GUI):

Developing a user-friendly interface would improve usability and allow for better interaction:

- **Graphical Representation**: The timetable can be displayed visually in a calendar-like format, showing schedules for students, instructors, or departments.

These extensions would make the timetable scheduler more flexible, user-friendly, and adaptable to complex scheduling needs.

CONCLUSION

The timetable scheduler successfully generates a valid timetable for seven courses for 3rd Year 2nd Sem in SRM University AP using GNU Prolog, modeling the problem as a CSP. The solution handles 21 theory classes and 4 lab sessions across 66 periods, satisfying complex constraints like CSE 437's time restrictions and SEC 137's consecutive hours. GNU Prolog's declarative programming and backtracking ensure correctness and efficiency, fitting within the 8 GB RAM system. The project demonstrates the power of logic programming in solving real-world scheduling problems, aligning with SRM's AI curriculum (Units 3 and 4) and supporting internship applications at Black Komodo Investments. Future enhancements could include room/professor assignments or integration with external systems, but the current solution meets all objectives as a standalone GNU Prolog application.