University Of Asia Pacific

Technical Report on

STUDENT MARKS EVALUATION SYSTEM

(A Prolog System To Evaluate Marks Using Rules And Backtracking)

Course Title: Artificial Intelligence & Expert System Lab
Course Code: CSE 404

Prepared for:
Bidita Sarkar Diba
Lecturer, Dept. of CSE
University of Asia Pacific

Prepared by:
Abdullah Al-Mamun
ID No: 22101158
Sec: D

Date: 11 August 2025

Department of Computer Science and Engineering
University of Asia Pacific

i. Problem Title

Student Performance Evaluation System in Prolog

ii. Problem Description

This project is a **Prolog-based Student Performance Evaluation System** designed to store, manage, and analyze academic data for students in an educational institution. The knowledgebase is implemented using logical facts and rules, enabling structured storage of information and automated reasoning to evaluate student performance.

The system primarily serves the following purposes:

1. **Data Storage & Organization**
    - Uses *facts* to store essential details, including:
        - Student names
        - Subjects offered
        - Marks obtained by each student in each subject
        - Attendance records (number of days present)
        - Number of assignments completed
2. **Rule-Based Performance Analysis**
    - Implements *rules* that process stored facts to generate insights such as:
        - **Pass/Fail Status**: Determines whether a student passes or fails a subject based on predefined marks criteria.
        - **Letter Grades**: Assigns grades (A, B, C, D, F) according to mark ranges.
        - **Total Marks**: Calculates the total marks obtained by a student across all subjects.
        - **Average Marks**: Computes the mean performance score for each student.
        - **Topper Identification**: Finds the student(s) with the highest average marks.
        - **Scholarship Eligibility**: Checks if a student meets criteria for scholarship consideration based on average marks and attendance.
        - **Weak Student Detection**: Flags students with consistently low marks or poor attendance for academic intervention.
3. **Benefits & Applications**

    - ✓ Facilitates **academic decision-making** for teachers and administrators.
    - ✓ Helps identify top performers for **awards and scholarships**.
    - ✓ Enables early detection of academically struggling students for **remedial action**.
    - ✓ Provides **flexible querying** — users can run targeted queries to extract specific performance metrics.

This knowledgebase ensures **structured, query-driven analysis** that is ideal for educational data management in logic programming environments.

iii. Tools and Languages Used

- **Programming Language: Prolog**

Prolog is a powerful logic programming language selected for this project due to its declarative nature, which allows knowledge to be represented elegantly through facts and rules. Its inherent strength lies in logical reasoning and pattern matching, making it especially suitable for modeling complex relationships and rules within the domain of student performance evaluation. Prolog's ability to infer conclusions from given data provides a robust foundation for implementing intelligent queries and decision-making processes in this knowledgebase.

- **Interpreter: SWI-Prolog**

SWI-Prolog is a widely adopted, free, and open-source Prolog environment favored in both educational and research communities. It offers a rich set of built-in predicates and libraries that facilitate efficient data manipulation, mathematical calculations, and logical inference. The interpreter's user-friendly interface and comprehensive debugging tools make it ideal for developing, testing, and maintaining complex logic-based systems like this student performance evaluation knowledgebase.

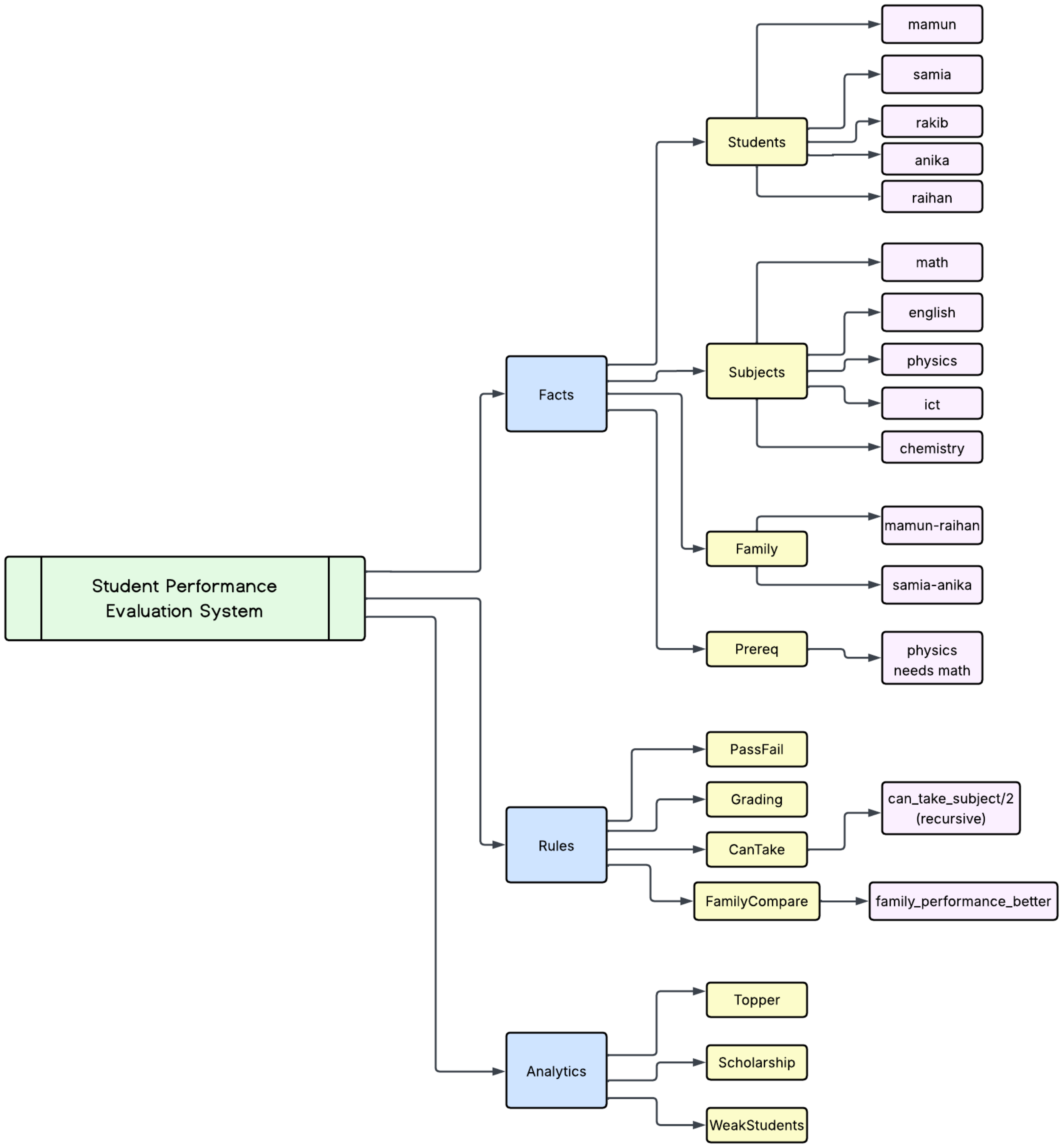
iv. Knowledgebase Structure Diagram

This hierarchical tree presents a comprehensive overview of student academic performance, structured based on a Prolog knowledgebase. Each student is represented as a primary node, branching into their respective subjects, where individual marks and grades are displayed to reflect their academic achievements clearly.

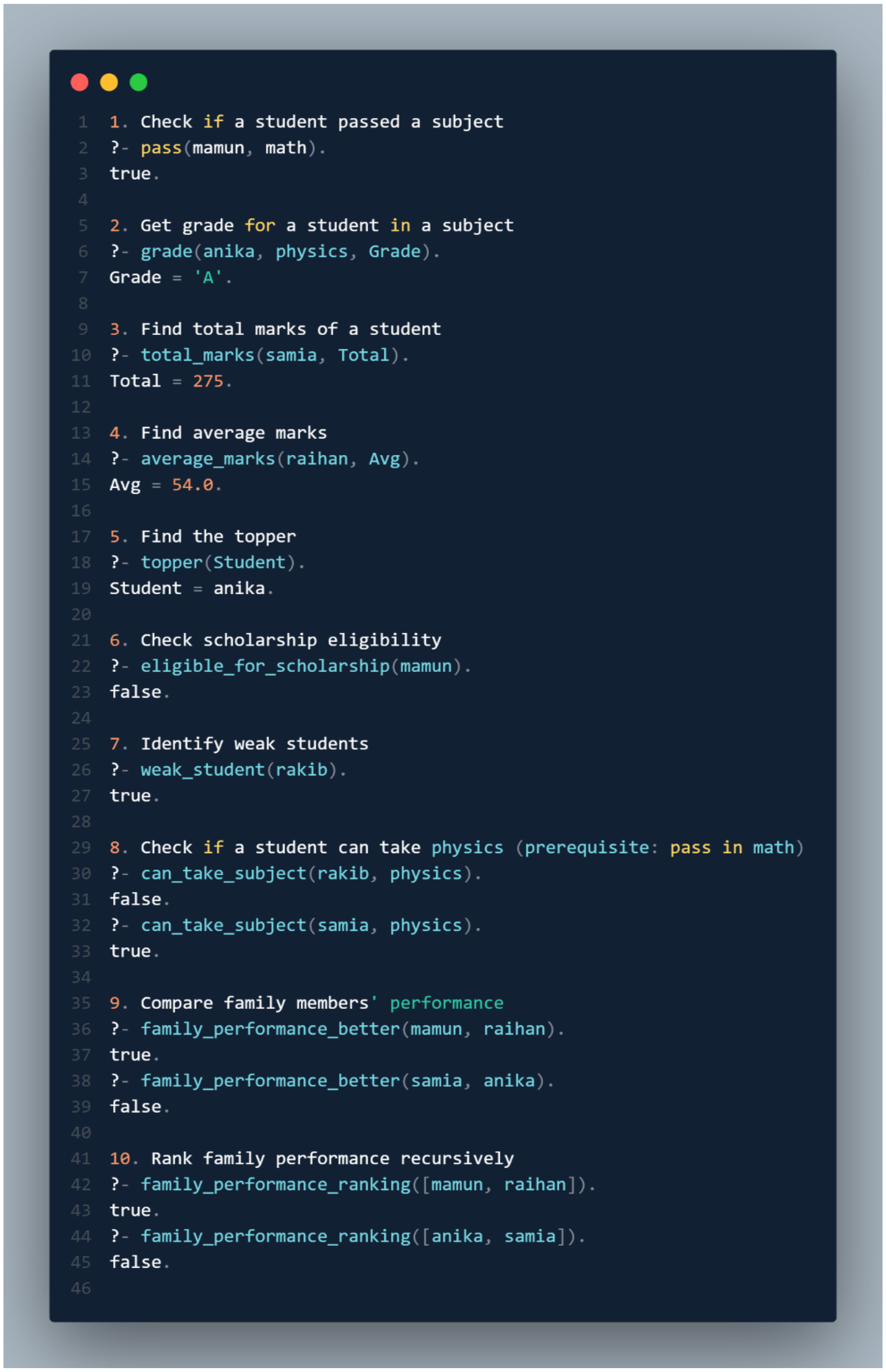
Beyond marks, the tree integrates additional dimensions such as attendance records and assignment completion, providing a holistic view of each student's engagement and consistency. Family relationships are also depicted, enabling performance comparisons among related students.

Furthermore, subject prerequisites, like the requirement to pass Mathematics before taking Physics, are incorporated to illustrate academic dependencies.

This visualization aids in identifying top performers, students eligible for scholarships, and those needing academic support, offering an insightful tool for educators and analysts alike.


Student Performance Evaluation System
Facts
Rules
Analytics
Students
Subjects
Family
Prereq
PassFail
Grading
CanTake
FamilyCompare
Topper
Scholarship
WeakStudents
mamun
samia
rakib
anika
raihan
math
english
physics
ict
chemistry
mamun-raihan
samia-anika
physics needs math
can_take_subject/2 (recursive)
family_performance_better

v. Sample Input/Output



```
1. Check if a student passed a subject
?- pass(mamun, math).
true.

2. Get grade for a student in a subject
?- grade(anika, physics, Grade).
Grade = 'A'.

3. Find total marks of a student
?- total_marks(samia, Total).
Total = 275.

4. Find average marks
?- average_marks(raihan, Avg).
Avg = 54.0.

5. Find the topper
?- topper(Student).
Student = anika.

6. Check scholarship eligibility
?- eligible_for_scholarship(mamun).
false.

7. Identify weak students
?- weak_student(rakib).
true.

8. Check if a student can take physics (prerequisite: pass in math)
?- can_take_subject(rakib, physics).
false.
?- can_take_subject(samia, physics).
true.

9. Compare family members' performance
?- family_performance_better(mamun, raihan).
true.
?- family_performance_better(samia, anika).
false.

10. Rank family performance recursively
?- family_performance_ranking([mamun, raihan]).
true.
?- family_performance_ranking([anika, samia]).
false.
```

vi. Conclusion and Challenges

Conclusion:
The **Student Performance Evaluation System in Prolog** successfully demonstrates how a logic programming approach can be applied to academic performance analysis. By representing data as facts and academic rules as logical predicates, the system can efficiently process complex queries such as determining pass/fail status, assigning letter grades, identifying top-performing students, and detecting underperforming students based on attendance and marks.

This approach ensures:

- **Clarity of Logic:** Rules for evaluation are explicitly defined in a declarative format, making reasoning transparent.
- **Flexibility in Querying:** Users can retrieve insights by asking specific questions, without needing to manually compute results.
- **Educational Utility:** This framework can serve as a foundation for institutions aiming to monitor student performance, recommend remedial measures, or identify scholarship candidates.

Overall, the system demonstrates that Prolog is a suitable choice for building intelligent, rule-based academic evaluation tools, providing both accuracy and scalability in performance analysis.

Challenges:

1. **Learning Curve for Prolog**
   - While Prolog is powerful for logical reasoning, it is less intuitive for users unfamiliar with declarative programming paradigms.
   - Non-technical educators or administrators may require additional training to formulate effective queries in Prolog syntax.
2. **Static Nature of Data**
   - The current system uses predefined facts stored directly in the code.
   - Real-world academic systems often require integration with dynamic databases or student management platforms, which would need additional code for real-time data updates.
3. **Hardcoded Evaluation Criteria**
   - Grading scales, attendance requirements, and scholarship eligibility thresholds are embedded in the source code.
   - Any modification (e.g., changing a pass mark from 40 to 50) requires altering and redeploying the code, rather than adjusting a configurable parameter.
4. **Limited User Interface**
   - The system currently operates in a command-line Prolog environment, which may not be user-friendly for non-technical stakeholders.
   - An intuitive front-end or web interface could make the system more accessible.