Discrete Structures

LAB I

JavaBased Logical Expression Evaluator and Inference Engine

Team members

جون وليم فوزي فهمي 22010701

أحمد راجي حسين 22011690

Problem Statement:

This project involved developing a two-part application using Java. The first part is a robust parser capable of handling logical expressions of propositional logic, validates them and evaluates their output after the user enters values.

The second part is an inference engine that can apply inference rules to logical expressions. The engine should take logical expressions as input, identify applicable inference rules, and generate the corresponding output based on the rules' logic.

Data Structures Used:

1. **Stacks** - Used in the conversion of logical expressions from infix to postfix notation and for evaluating these postfix expressions.
2. **Arrays** - Employed to store sequences of tokens or characters in logical expressions.
3. **Hashmaps** - Utilized to map user input symbols to standardized letters (P, Q, R, S)

Implementation Details:

Part 1: Logical Expressions Solver

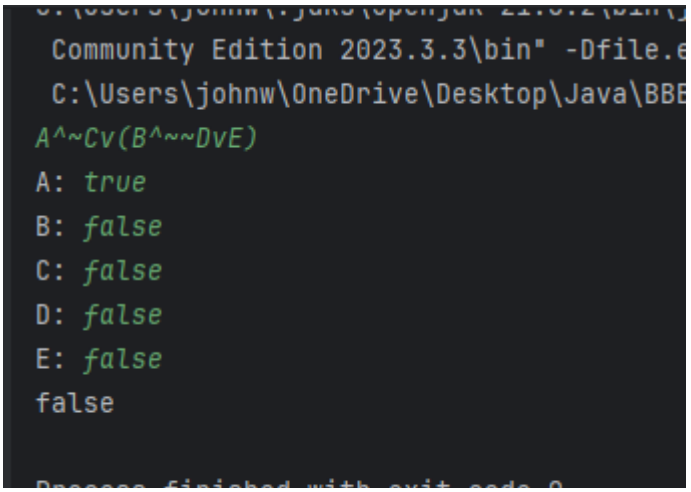
- Implements a parser that supports simplified propositional logic syntax including negation, conjunction, disjunction, and implication.
- The parser first validates the expression and normalises it to a standard form by removing spaces and handling regex expressions.
- The parser handles parentheses for grouping and follows the precedence of operators.
- It converts logical expressions to postfix notation using a stack, evaluates them with another stack based on user-provided Boolean values, and outputs the result.

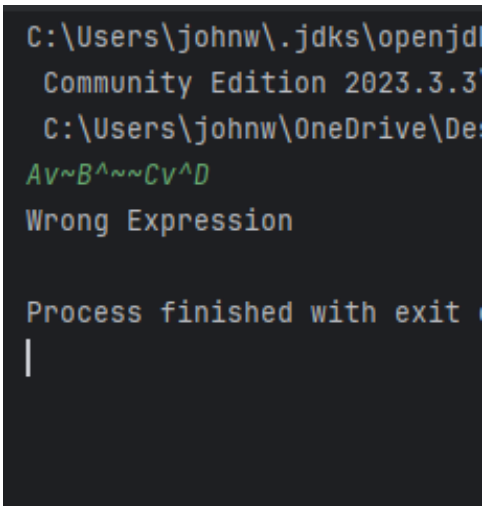
Part 2: Inference Engine

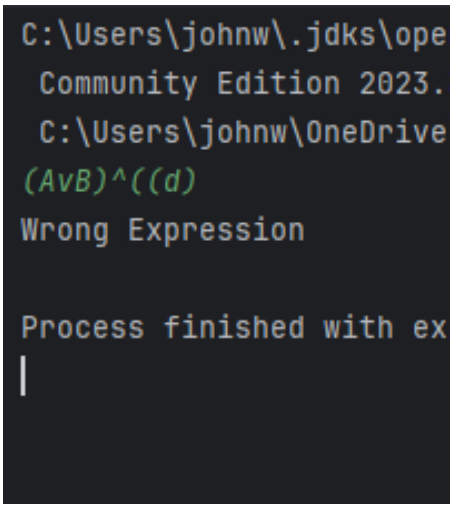
- Dynamically loads inference rules from a text file, enabling easy updates and rule additions.
- Standardizes user input by mapping all symbols to the default letters P, Q, R, S using hashmaps and the replaceAll() method.
- Iteratively compares the user input against all loaded rules using arrays, linked lists, to find applicable inference rules.

Sample Runs and Test Cases:

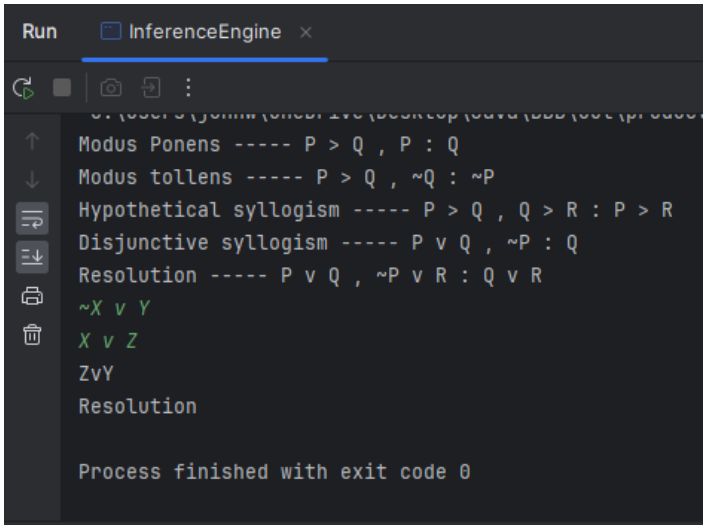
1. Logical Expression Evaluation:

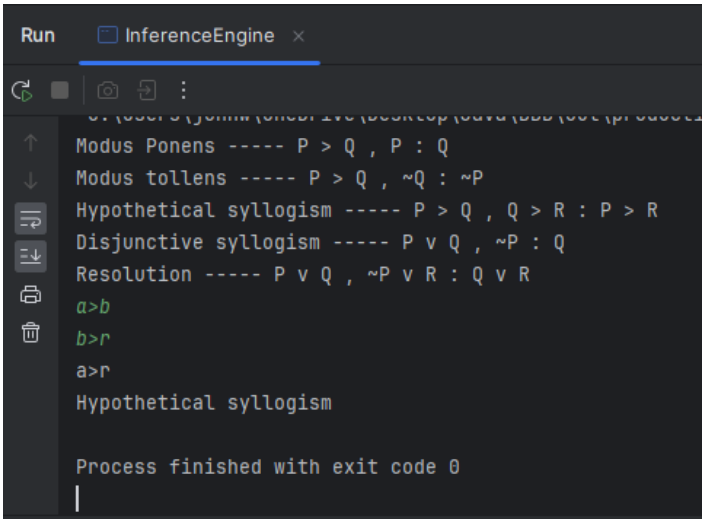


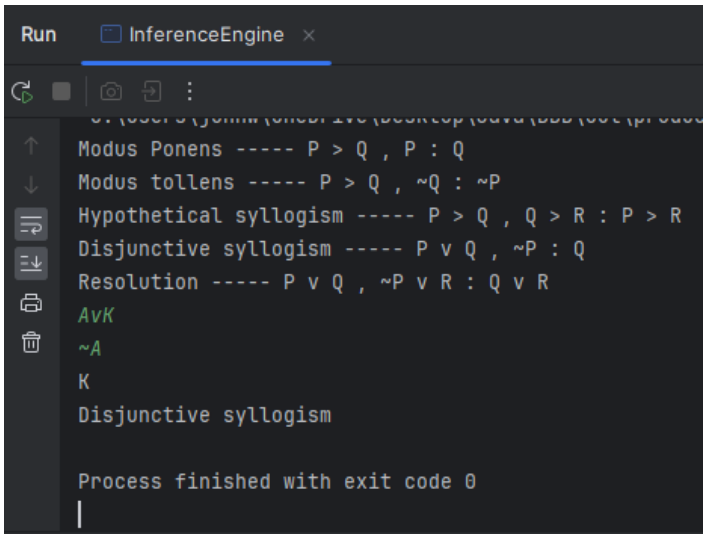


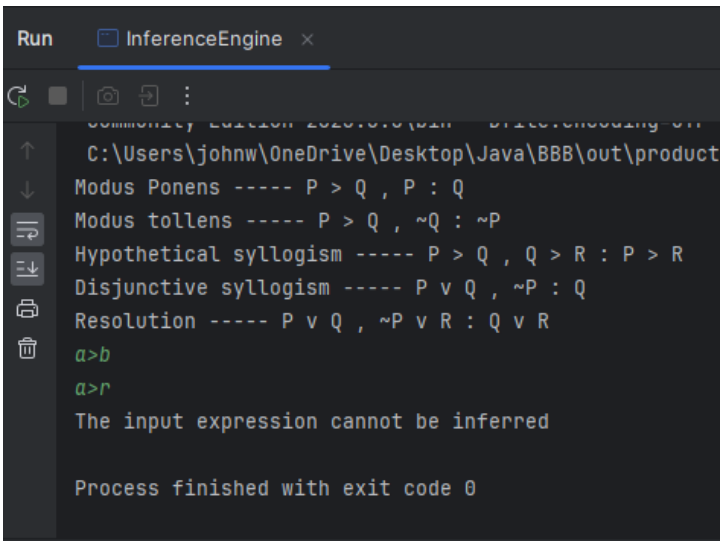


2. Inference Rule Application:









Assumptions:

- **Part 1:**
  - Reasons for a wrong expression are
    - Not symmetric parentheses
    - Operand consists of two or more characters (operand must be one letter)
    - Two operators following each other(excluding the ~ operator)
  - Negating operators can be repeated where each (“~~”) is replaced with (“”)
  - Expression can be in uppercase or lowercase or mixed
  - Spaces between characters don’t matter
- **Part 2:**
  - Rules are written into a text file in the order of

    Rule Name

    Exp1 , exp2 : inference

  - All Rules must be written used the 4 characters {P,Q,R,S} and used in the same order according to their order of occurence.
  - All rule letters must be in upper case
  - Spaces in rules don’t matter
  - User must input correct expression form with correct characters, letter casing matters but spacing doesn’t matter

Additional Details:

- The application includes detailed error handling and validation mechanisms.
- The system is designed to be scalable with the potential to support more complex logical operations and additional inference rules in future developments.

Conclusion:

This project successfully demonstrates the use of Java and various data structures to implement a logical expression solver and an inference engine. The application efficiently handles logical operations and rule-based inference, providing a robust tool for educational or practical use in the field of discrete mathematics.