

# Discrete Mathematics Toolkit

Philopateer Reda, Kareem Gamal Elsaid, Fahmy Ashraf Fahmy, Mohamed Naser Farhat, Mahmood Samir Mahmood

## Conclusion & Call to Action

# Discrete Mathematics Toolkit: Explore the Beauty of Mathematics

The Discrete Mathematics Toolkit empowers students to grasp complex ideas and helps educators inspire mathematical thinkers. Join us in transforming discrete math education.



**Learn Foundations**

**Visualize Concepts**

**Apply Innovation**

**Empower Future**

[Try the Toolkit](#)

[Learn More](#)

[Get Involved](#)

Made with **GAMMA**

# Crafted for Clarity and Comfort

Our design prioritizes an intuitive and visually engaging experience, ensuring focus and ease of use.



## Professional Dark Mode

Deep theme with vibrant cyan accents reduces eye strain and enhances focus.



## Intuitive Feedback

Clear, color-coded indicators (green for success, red for errors) provide instant guidance.



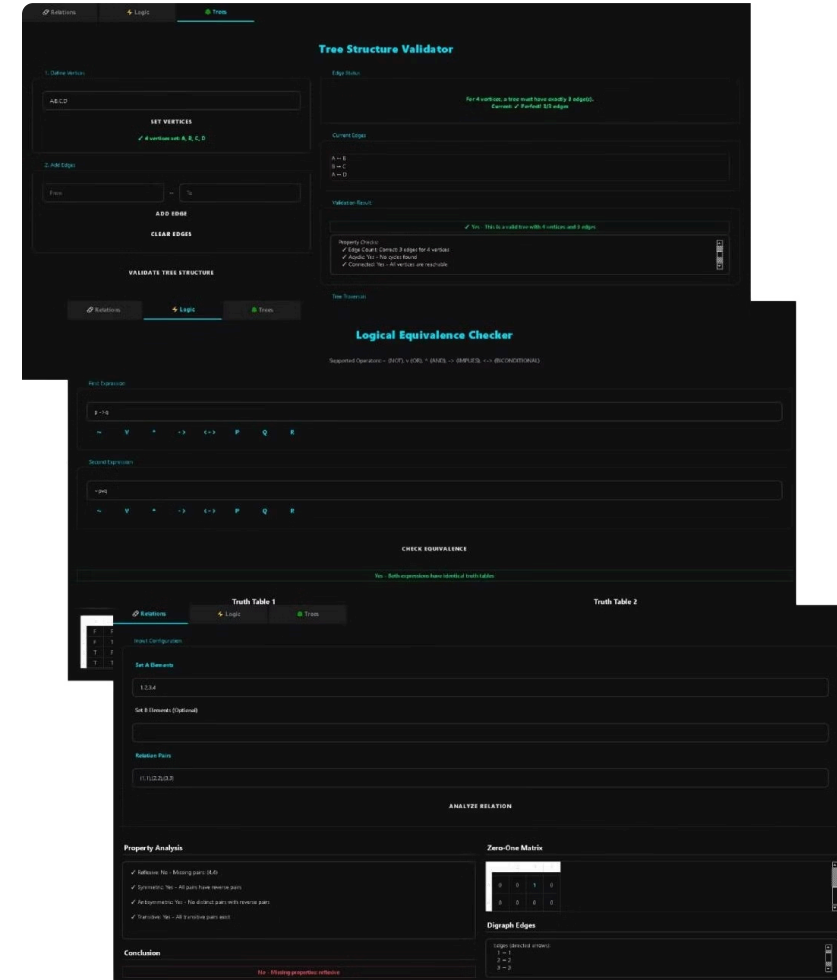
## Seamless Responsiveness

Smooth hover effects and tabbed navigation adapt to any screen size.



## High DPI Support

Crystal-clear visuals and sharp text, optimized for 4K and Retina displays.



# Tree Validator: Interactive Graph Theory Tool

Explore graph theory with the Tree Validator, ensuring your structures meet tree definitions.



## Intuitive Graph Construction

Easily input vertices and edges to construct your desired graph or tree.



## Powerful Validation Algorithms

Uses DFS, BFS, and edge counting to verify tree properties.



## Instant, Clear Feedback

Get instant visual confirmation if your structure is a valid tree, or see reasons for invalidity.

# Tree Analysis: Code & Interface

Explore our intuitive interface and the robust Python code behind it.

## Interactive Interface:

Relations

Logic

Trees

1. Define Vertices

A,B,C,D

SET VERTICES

✓ 4 vertices set: A, B, C, D

2. Add Edges

From To

ADD EDGE

CLEAR EDGES

VALIDATE TREE STRUCTURE

Tree Structure Validator

Edge Status

For 4 vertices, a tree must have exactly 3 edge(s).  
Current: ✓ Perfect! 3/3 edges

Current Edges

A → B  
B → C  
A → D

Validation Result

✓ Yes - This is a valid tree with 4 vertices and 3 edges

Property Checks:  
✓ Edge Count: Correct: 3 edges for 4 vertices  
✓ Acyclic: Yes - No cycles found  
✓ Connected: Yes - All vertices are reachable

Tree Traversals

Root Vertex: A

➤ Pre-order: A, B, C, D  
➤ In-order: C, B, A, D  
➤ Post-order: C, B, D, A

## Code:

```
visited = set()

def dfs(node, parent):
    visited.add(node)
    for neighbor in self.adjacency[node]:
        if neighbor not in visited:
            if dfs(neighbor, node):
                return True
        elif neighbor != parent:
            # Found a back edge (cycle)
            return True
    return False

# Check from any starting vertex
for start in self.vertices:
    if start not in visited:
        if dfs(start, None):
            return (True, "Cycle detected")

return (False, "No cycles found")
```

# Logic Equivalence Checker: Propositional Logic Engine

Dive into the core of propositional logic with our powerful Equivalence Checker.



## Expression Parsing

Supports complex expressions with operators like  $\sim$  (NOT),  $\vee$  (OR),  $\wedge$  (AND),  $\rightarrow$  (IMPLIES), and  $\leftrightarrow$  (BICONDITIONAL).



## Truth Tables

Automatically generate comprehensive truth tables for any propositional logic expression, making complex logic transparent.



## Equivalence Testing

Verify logical equivalences, such as De Morgan's Laws, to confirm if two expressions yield identical truth values.



## User-Friendly Input

Input expressions effortlessly with quick-insert buttons for all logical operators, enhancing speed and reducing errors.

# Logic Equivalence Checker: Code & Interface

Explore our intuitive interface and the robust Python code behind it.

## Interactive Interface:

Relations

Logic

Trees

Logical Equivalence Checker

Supported Operators: ~ (NOT), v (OR), ^ (AND), -> (IMPLIES), <-> (BICONDITIONAL)

First Expression

p -> q

~ v ^ -> <-> P Q R

Second Expression

~ p v q

~ v ^ -> <-> P Q R

CHECK EQUIVALENCE

Yes - Both expressions have identical truth tables

Truth Table 1

P	Q	p -> q
F	F	T
F	T	T
T	F	F
T	T	T

Truth Table 2

P	Q	~ p v q
F	F	T
F	T	T
T	F	F
T	T	T

## Code:

```
variables = sorted(self._extract_variables(expression))

if not variables:
    # No variables, just evaluate
    try:
        result = self._evaluate(expression, {})
        return ([], [{}, result])
    except:
        raise ValueError("Invalid expression with no variables")

rows = []
# Generate all combinations of True/False for variables
for values_tuple in product([False, True], repeat=len(variables)):
    values_dict = dict(zip(variables, values_tuple))
    result = self._evaluate(expression, values_dict)
    rows.append((values_dict, result))

return (variables, rows)
```

# Relation Checker: Analyze Mathematical Relations

Our Relation Checker provides powerful tools to analyze mathematical relations with depth and clarity.



## Property Verification

- Test Reflexive, Symmetric, Antisymmetric
- Verify Transitive properties
- Ensure consistency and validity



## Equivalence Relations

- Detect equivalence relations
- Partition sets into equivalence classes
- Simplify complex structures



## Dynamic Visualizations

- Generate Zero-One Matrices
- Display Directed Graphs
- Intuitive visual aids



# Relation Checker: Code & Interface

Explore our intuitive interface and the robust Python code behind it.

## Interactive Interface:

The interface is a web-based application for checking relation properties. It features a dark theme and a sidebar with navigation tabs: 'Relations' (selected), 'Logic', and 'Trees'. The main area is divided into sections for input and analysis.

**Input Configuration:**

- Set A Elements:** A text input field containing '1,2,3,4'.
- Set B Elements (Optional):** An empty text input field.
- Relation Pairs:** A text input field containing '(1,1),(2,2),(3,3)'.

**ANALYZE RELATION**

**Property Analysis:**

- ✓ Reflexive: No - Missing pairs: (4,4)
- ✓ Symmetric: Yes - All pairs have reverse pairs
- ✓ Antisymmetric: Yes - No distinct pairs with reverse pairs
- ✓ Transitive: Yes - All transitive pairs exist

**Zero-One Matrix:**

0	0	1	0
0	0	0	0

**Digraph Edges:**

Edges (directed edges):

- 1 ~ 1
- 2 ~ 2
- 3 ~ 3

**Conclusion:**

No - Missing properties: reflexive

## Code:

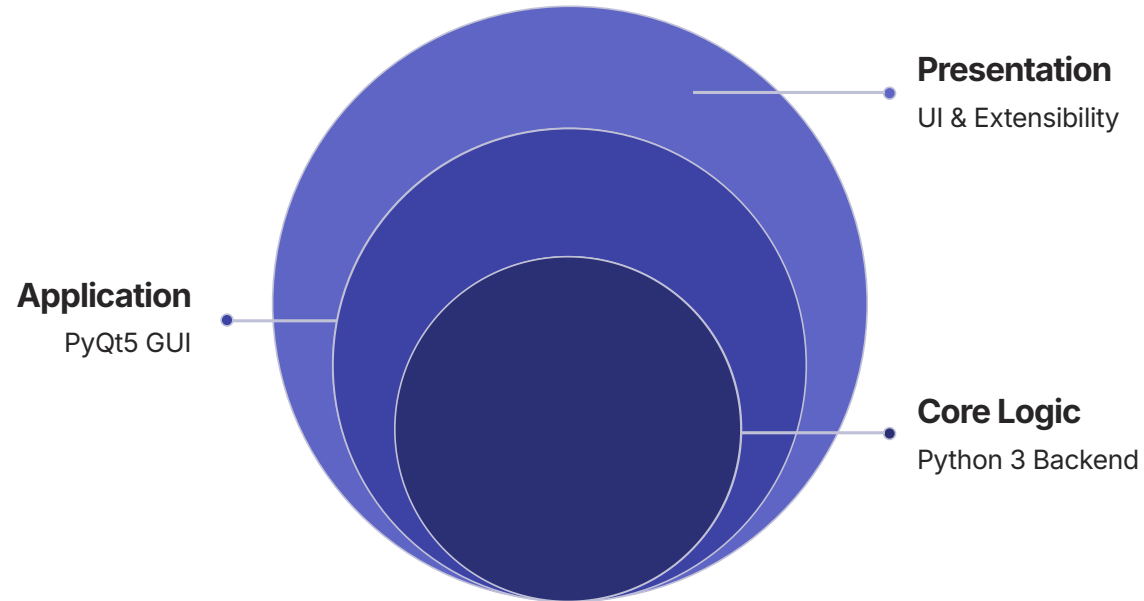
```
violations = []
for a, b in self.relation:
    for b2, c in self.relation:
        if b == b2 and (a, c) not in self.relation:
            violations.append((a, b, c))

is_transitive = len(violations) == 0
explanation = "Yes - All transitive pairs exist" if is_transitive else \
    f"No - Missing: {' '.join(f'({a},{c})' for a, b, c in violations[:3])}"

result = (is_transitive, explanation)
self._cache['transitive'] = result
return result
```

# Technical Architecture

Our toolkit is **Built for Performance and Maintainability**, leveraging robust technologies for a seamless user experience.



## Tech Stack

- **Language:** Python 3 (scripting, algorithms).
- **GUI Framework:** PyQt5 (interactive, cross-platform UI).
- **Architecture:** Separated backend/frontend.

## Structure

- **Backend:** Python logic (algorithms, data processing).
- **Frontend:** Intuitive PyQt5 widgets.
- **Extensibility:** Modular for new features.

# What's Next: Expanding Possibilities

Our roadmap includes exciting new features to enhance your discrete mathematics learning and teaching.



## Set Theory Module

Explore fundamental set operations with interactive tools.



## Graph Visualization

Visually plot and analyze complex graphs and trees with customizable layouts.



## Export Capabilities

Save analysis and visualizations to PDF or LaTeX for easy sharing.



**Thank You**