**Department of Computer Engineering and Software Systems**
**Faculty of Engineering, Ain Shams University**

**CSE312 | Electronic Design Automation**

# TRUTH

Presnted By:

Abdelrahman Mostafa Sallam | 22P0150
Mohamed Ashraf Mohamed | 22P0210
Marwan Ahmed Elsahy | 22P0201
Andrew Rami Bassily | 22P0187
Habeba Adel Sallam | 22P0259
Saif El Din Wael | 22P0191

Presented To:
Dr. Eman
Eng. Abdelrahman

Electronic Design Automation
Group 3

# TABLE OF CONTENTS

# *LIST OF FIGURES*

# 1. INTRODUCTION TO RO|BDD

**Abstract**

In computer science and digital logic design, Binary Decision Diagrams (BDDs) are a way to represent and analyze Boolean functions efficiently. Boolean functions are expressions made up of binary variables (true/false, 1/0) combined using logical operators like AND, OR, and NOT. These functions are essential in areas like digital circuit design, software verification, and artificial intelligence.

Reduced Ordered Binary Decision Diagrams (ROBDDs) are a simplified and optimized form of BDDs. They eliminate redundancy and follow strict ordering rules for variables, making them more compact and efficient for computation, where our main goal is to check equivalence between the ROBDD of the 2 functions.

## 1.1 DEFINING A BDD

### Binary Decision Diagrams (BDD)

1. **Definition**: A BDD is a graph-based representation of a Boolean function. It is a binary tree-like structure where:
    - Each non-terminal node represents a variable.
    - Each edge represents a possible value (0 or 1) of that variable.
    - Terminal nodes are either 0 or 1, representing the function's output for a specific combination of variable values.

2. **Structure**:
    - **Nodes**: Represent decision points for variables.
    - **Edges**: Show the path for each possible variable value (low for 0, high for 1).
    - **Terminal Nodes**: Contain the result of the function.

3. **Construction**: BDDs are built by breaking down the Boolean function into smaller sub-functions. Each decision leads to further branches until all variables are evaluated.

4. **Advantages**:
    - Clear and visual representation of Boolean functions.
    - Useful for simplifying and analyzing complex functions.

5. **Disadvantages**:
    - Can grow exponentially in size for some functions, making them inefficient.

## 1.2 DEFINING A ROBDD

### Reduced Ordered Binary Decision Diagrams (ROBDD)

1. **Definition**: An ROBDD is a compact version of a BDD. It uses two rules to simplify the graph:
    - **Variable Ordering**: Each variable appears in the same order along all paths.
    - **Elimination of Redundancy**: Identical subgraphs and redundant nodes are merged or removed.

2. **Optimization Techniques**:
    - **Merging**: If two nodes have the same variable, low branch, and high branch, they are merged into one.
    - **Skipping**: If a node's low and high branches point to the same node, it is removed, and its parent is directly connected to the common child.

3. **Benefits of ROBDD**:
    - Significantly reduces memory usage by eliminating redundancy.
    - Ensures a unique representation for each Boolean function given a variable order.

4. **Applications**:
    - **Digital Circuit Design**: Used to verify the equivalence of different circuit designs.
    - **Software Verification**: Ensures that programs meet their specifications.
    - **Artificial Intelligence**: Optimizes decision-making processes.

### Example

Consider the Boolean function f(a,b)=a AND bf(a, b) = a \text{ AND } bf(a,b)=a AND b. Its BDD representation includes:

- A decision node for aaa with two branches: a=0a = 0a=0 and a=1a = 1a=1.
- A further decision for bbb, leading to terminal nodes 000 and 111 based on the evaluation of aaa and bbb.

When reduced into a ROBDD:

- Redundant nodes are merged.
- The resulting graph is minimal and uniquely represents the function.

## 1.3 DESIGN SPECIFICATIONS

In this section we will propose a broad overview of the system's context, design principles as well as functionality.

### 1.3.1 INTRODUCTION

This application uses **Python's libraries** to build a **Graphical User Interface (GUI)** that visualizes **Binary Decision Diagrams (BDD)** and **Reduced Ordered Binary Decision Diagrams (ROBDD)**. It helps users understand how Boolean expressions are represented using BDDs and how ROBDDs optimize them.

The program **includes**:

- Inputting Boolean expressions.
- Building BDDs and ROBDDs.
- Displaying the results interactively in a GUI.

### 1.3.2 KEY CONCEPTS

**Boolean Expression:**

A mathematical expression using variables, logical operators (AND, OR, NOT), and binary values (0 and 1).

- Example: A AND B

**BDD (Binary Decision Diagram):**

A tree-like structure representing Boolean functions. Each node represents a variable, and branches represent the binary values (0 and 1).

**ROBDD (Reduced Ordered BDD):**

An optimized version of BDD where:

- Variables are ordered consistently across paths.
- Duplicate nodes are removed.

**Graph Representation:** The diagrams are displayed in two ways:

### 1.3.3 USER INTERFACE DESIGN

**Input Section:** Text box for Boolean expression and variable list, where the user inputs whatever expression he likes then he selects the variables order, if not selected then it's done alphabetically.

**Buttons:**

- Build BDD
- Build ROBDD
- Draw on Canvas
- Visualize with NetworkX Python Library

# 2. GENERAL TABLES

## 2.1 PARAMETERS TABLE

| Parameter | Type | Description |
|---|---|---|
| **expression** | String | Boolean expression |
| **variables** | List | List of variables used in the expression |
| **assignment** | Dict | Variable assignments for evaluation |
| **NODE_RADIUS** | Integer | Radius of regular decision nodes in Canvas |
| **TERMINAL_RADIUS** | Integer | Radius of terminal nodes in Canvas |
| **VERTICAL_SPACING** | Integer | Vertical spacing between nodes in Canvas |
| **HORIZONTAL_SPACING** | Integer | Horizontal spacing between nodes in Canvas |
| **EDGE_TEXT_OFFSET_X** | Integer | Horizontal offset for edge labels |
| **EDGE_TEXT_OFFSET_Y** | Integer | Vertical offset for edge labels |

## 2.2 I/O TABLE

| Input | Action | Output |
|---|---|---|
| **Boolean expression** | Build BDD button clicked | BDD built and displayed. |
| **Boolean expression** | Build ROBDD button clicked | ROBDD built and displayed |
| **Valid Boolean expression** | Draw button clicked | Diagram on Canvas |
| **Valid Boolean expression** | Visualize button clicked | Graph plotted with |

## 2.3 CORE FUNCTIONS TABLE

| Function | Purpose |
|---|---|
| **build_bdd()** | Constructs the BDD tree |
| **build_robdd()** | Optimizes the BDD to ROBDD |
| **evaluate_expression()** | Evaluates Boolean expressions based on variable assignments |
| **draw_bdd()** | Draws BDD on the GUI canvas |
| **create_networkx_graph()** | Creates a graph representation of the ROBDD |

## 3. TECHNICAL CODE WALKTHROUGH

### 3.1 CODE LIBRARIES

```python

import tkinter as tk
from tkinter import messagebox
import re
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

**FIGURE 1 LIBRARIES**

#### import tkinter as tk

- **Purpose:** Imports the Tkinter library for creating graphical user interfaces (GUIs).
- **Example Use:** tk.Button, tk.Label, and tk.Entry widgets for building GUI applications.

#### from tkinter import messagebox

- **Purpose:** Imports the messagebox module from Tkinter, which provides pop-up dialog boxes for showing messages, warnings, or errors.
- **Example Use:** messagebox.showinfo("Title", "This is an info message").

#### import re

- **Purpose:** Imports Python's regular expression library for string pattern matching and manipulation.
- **Example Use:** re.match, re.search, and re.findall for text validation

#### import networkx as nx

- **Purpose:** Imports NetworkX, a library for creating, analyzing, and visualizing complex networks (graphs).
- **Example Use:** Creating graphs, adding nodes and edges, and analyzing network properties.

#### import matplotlib.pyplot as plt

- **Purpose:** Imports Matplotlib's plotting library for creating static, animated, and interactive visualizations.
- **Example Use:** plt.plot() or plt.show() to display graphs and charts.

#### from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

- **Purpose:** Integrates Matplotlib plots into a Tkinter GUI.
- **Example Use:** Embedding a Matplotlib plot as a widget in a Tkinter window using FigureCanvasTkAgg

## 3.2 CLASS BDDNODE

```python
class BDDNode:
    def __init__(self, var=None, low=None, high=None, value=None):
        self.var = var
        self.low = low
        self.high = high
        self.value = value

    def is_terminal(self):
        return self.value is not None

    def __eq__(self, other):
        if not isinstance(other, BDDNode):
            return False
        return (self.var == other.var and
                self.low == other.low and
                self.high == other.high and
                self.value == other.value)

    def __hash__(self):
        return hash((self.var, self.low, self.high, self.value))
```

FIGURE 2 BDDNODE

### 1. Init constructor

**var:** Represents the variable for the current node. In the context of a BDD, this is usually a Boolean variable
**low:** Points to the low child node (the "false" branch in the decision diagram).
**high:** Points to the high child node (the "true" branch in the decision diagram).
**value:** Holds the value of the node, which could be True or False if it's a terminal node.

## 2.Functions

- **The is_terminal_method** to check if the node (is terminal), meaning it holds a Boolean value (True or False).
- **The __eq__ method** compares two nodes for equality by checking their var, low, high, and value attributes.
- **The __hash__ method** generates unique hash for a node based on these attributes, allowing it to be used in hash-based collections like sets or dictionaries.
- This class is fundamental for building and manipulating **BDDs**, which represent Boolean functions.

## 3.3 CLASS BDDBUILDER

This is the main part of the technical code and it contains all the functions and attributes as well as variables used for building BDD diagram and building ROBDD diagram.

```python
class BDDBuilder:
    def __init__(self):
        self.node_cache = {}
        self.NODE_RADIUS = 10  # Even smaller node radius
        self.TERMINAL_RADIUS = 8  # Even smaller terminal radius
        self.VERTICAL_SPACING = 80 # Even smaller vertical spacing
        self.HORIZONTAL_SPACING = 20  # Even smaller horizontal spacing
        self.EDGE_TEXT_OFFSET_X = 5 # Even smaller text offset
        self.EDGE_TEXT_OFFSET_Y = 2 # Even smaller text offset

    def build_bdd(self, expression, variables, assignment={}):
        if not variables:
            value = self.evaluate_expression(expression, assignment)
            return BDDNode(value=value)

        current_var = variables[0]
        assignment_false = assignment.copy()
        assignment_false[current_var] = 0
        assignment_true = assignment.copy()
        assignment_true[current_var] = 1

        low_branch = self.build_bdd(expression, variables[1:], assignment_false)
        high_branch = self.build_bdd(expression, variables[1:], assignment_true)

        node = BDDNode(var=current_var, low=low_branch, high=high_branch)
        return node

    def build_robdd(self, bdd_root):
        if bdd_root is None:
            return None

        if bdd_root.is_terminal():
            if bdd_root.value not in self.node_cache:
                self.node_cache[bdd_root.value] = bdd_root
            return self.node_cache[bdd_root.value]

```

**FIGURE 3 CLASS BDDBUILDER #1**

```python
        bdd_root.low = self.build_robdd(bdd_root.low)
        bdd_root.high = self.build_robdd(bdd_root.high)

        if bdd_root.low == bdd_root.high:
            return bdd_root.low

        if bdd_root in self.node_cache:
            return self.node_cache[bdd_root]

        self.node_cache[bdd_root] = bdd_root
        return bdd_root


    def evaluate_expression(self, expression, assignment):
        try:
            eval_expr = expression
            for var, val in assignment.items():
                eval_expr = eval_expr.replace(var, str(val))
            return int(bool(eval(eval_expr)))

        except Exception as e:
            raise ValueError(f"Invalid expression evaluation: {e}")


    def draw_bdd(self, canvas, root, x, y, dx=50, level=0):
        if root is None:
            return

        if root.is_terminal():
            if root.value == 0:
                canvas.create_oval(x - self.TERMINAL_RADIUS, y - self.TERMINAL_RADIUS,
                                   x + self.TERMINAL_RADIUS, y + self.TERMINAL_RADIUS, fill="red")
            else:
                canvas.create_oval(x - self.TERMINAL_RADIUS, y - self.TERMINAL_RADIUS,
                               x + self.TERMINAL_RADIUS, y + self.TERMINAL_RADIUS, fill="lightgreen")
```

**FIGURE 4 CLASS BDDBUILDER #2**

```python
                canvas.create_text(x, y, text=str(int(root.value)))
                return

        canvas.create_oval(x - self.NODE_RADIUS, y - self.NODE_RADIUS, x + self.NODE_RADIUS,
                           y + self.NODE_RADIUS, fill="lightblue")
        canvas.create_text(x, y, text=root.var)

        next_y = y + self.VERTICAL_SPACING
        if root.low:
            canvas.create_line(x, y + self.NODE_RADIUS, x - dx, next_y - self.NODE_RADIUS, arrow=tk.LAST, dash=(2, 2))
            text_x = x - dx/2
            text_y = (y+self.NODE_RADIUS+ next_y - self.NODE_RADIUS)/2
            canvas.create_text(text_x - self.EDGE_TEXT_OFFSET_X, text_y - self.EDGE_TEXT_OFFSET_Y, text="0")
            self.draw_bdd(canvas, root.low, x - dx, next_y, dx // 2, level + 1)
        if root.high:
            canvas.create_line(x, y + self.NODE_RADIUS, x + dx, next_y - self.NODE_RADIUS, arrow=tk.LAST)
            text_x = x + dx / 2
            text_y = (y + self.NODE_RADIUS + next_y - self.NODE_RADIUS) / 2
            canvas.create_text(text_x + self.EDGE_TEXT_OFFSET_X, text_y - self.EDGE_TEXT_OFFSET_Y, text="1")
            self.draw_bdd(canvas, root.high, x + dx, next_y, dx // 2, level + 1)


    def create_networkx_graph(self, root, graph_type):
        graph = nx.DiGraph()
        node_queue = [(root, 0)]  # Include the level as part of the queue
        visited = {}

        while node_queue:
            node, level = node_queue.pop(0)

            if node in visited:
```

**FIGURE 5 CLASS BDD BUILDER #3**

```python
                continue
            visited[node] = level

            # Determine the node color (blue for regular, red or green for terminal nodes)
            if node.low is None and node.high is None:  # Terminal node
                color = 'red' if node.value == 0 else 'green'
                label = str(node.value)  # Explicitly set label as "0" or "1"
            else:
                color = 'lightblue'
                label = node.var if node.var else str(node.value)

            # Add the node with color and level as attributes
            graph.add_node(
                node,
                subset_key=level,
                label=label,  # Assign label for nodes
                color=color
            )

            if node.low is not None:
                # Add an edge for the low branch
                graph.add_edge(node, node.low, label='0', color='red')
                node_queue.append((node.low, level + 1))

            if node.high is not None:
                # Add an edge for the high branch
                graph.add_edge(node, node.high, label='1', color='green')
                node_queue.append((node.high, level + 1))

        return graph
```

FIGURE 6 CLASS BDD BUILDER #4

# A. Methods Break down

## "The __init__ method":

initializes an instance of the BDDBuilder class. It sets up various parameters used for building and drawing Binary Decision Diagrams (BDDs):

- **NODE_CACHE**: A dictionary used to store nodes that have already been created, allowing for optimization by reusing them when needed.
- **NODE_RADIUS**: Defines the radius of a standard node in the diagram.
- **TERMINAL_RADIUS**: Defines the radius of terminal nodes (0 or 1).
- **VERTICAL_SPACING**: The vertical distance between nodes when drawn.
- **HORIZONTAL_SPACING**: The horizontal distance between nodes when drawn.
- **EDGE_TEXT_OFFSET_X and EDGE_TEXT_OFFSET_Y**: These control the positioning of the labels on the edges between nodes.

## “Build_bdd Method”:

The build_bdd method recursively constructs a Binary Decision Diagram (BDD) based on a given Boolean expression and a list of variables. It works as follows:

- **Base Case**: If there are no more variables to process, it evaluates the expression using the current assignment of variable values and creates a terminal node (either 0 or 1).

- **Recursive Case**: For each variable, it creates a node and recursively builds two branches:
  - The **low_branch** corresponds to the case where the variable is assigned a value of 0.
  - The **high_branch** corresponds to the case where the variable is assigned a value of 1.

Each node in the BDD corresponds to a variable, and the edges between nodes represent the two possible outcomes for each variable (0 or 1). The method returns the root node of the BDD.

## “Build_robdd Method”:

The build_robdd method optimizes the BDD by reusing previously created nodes through memorization. The method works as follows:

- **Base Case**: If the node is terminal (it contains a value), it either returns the node from the cache if it has already been created, or it creates and caches it.

- **Recursive Case**: For non-terminal nodes, it recursively builds the low and high branches. If both branches are the same, it eliminates the redundancy by returning just one branch. The method ensures that each node is created only once, which reduces the size of the final diagram.

Optimization ensures that only necessary nodes are created, which results in a more compact BDD.

## “Evaluate_expression Method”:

The evaluate_expression method evaluates a given Boolean expression with a set of variable assignments. It works as follows:

- **Substitutes Variables**: The method replaces the variables in the expression with their corresponding values from the assignment.

- **Evaluates the Expression**: The modified expression is evaluated using Python's eval() function, which computes the Boolean result (0 or 1) based on the current variable assignments.

- **Error Handling**: If the expression is invalid, an exception is raised, indicating an error in the evaluation process.

This method allows for the dynamic evaluation of Boolean expressions based on different variable assignments.

## “Draw_bdd Method”:

The draw_bdd method is responsible for visually drawing the BDD using a canvas. It constructs the diagram by placing circles for nodes and lines for edges. The method works as follows:

- **Terminal Nodes**: If the node is terminal (contains a value of 0 or 1), it is represented as a circle (red for 0 and green for 1) with a label showing its value.

- **Non-terminal Nodes**: Non-terminal nodes are drawn as light blue circles, labeled with the corresponding variable.

- **Branching**: The method creates lines representing the low (0) and high (1) branches. These lines are drawn with arrows, and the labels "0" and "1" are placed near the edges to indicate the outcome of each branch.

- **Recursive Drawing**: The method recursively calls itself to draw the branches, adjusting the positions of the nodes to ensure a clear and organized diagram.

Visual representation is crucial for understanding the structure of the BDD and how the Boolean function is evaluated.

## "Create_networkx_graph Method":

The create_networkx_graph method converts the BDD structure into a directed graph using the NetworkX library. It creates nodes and edges for the graph and assigns attributes to them. The method works as follows:

- **Node Properties**: Each node is added to the graph with the following attributes:
  - subset_key: Represents the level of the node in the BDD (used for visualization).
  - label: The label for the node, either the variable name or the terminal value (0 or 1).
  - color: The color of the node (light blue for regular nodes, red for terminal 0, and green for terminal 1).

- **Edge Properties**: Edges are added between nodes to represent the transitions between low and high branches. Each edge is labeled with "0" or "1" to indicate the outcome of the branch.

- **Breadth-First Traversal**: The method uses a breadth-first traversal (BFS) to visit each node in the BDD. It ensures that all nodes and edges are included in the graph.

- **Memorization**: A visited set is used to avoid revisiting nodes, ensuring that each node is processed only once.

# B. Table of Methods

| Method | Description |
|---|---|
| __init__ | Initializes an instance of BDDBuilder. Sets up default parameters such as node_cache, node and terminal radius, and spacing for drawing |
| build_bdd | Recursively builds a Binary Decision Diagram (BDD) from a Boolean expression and a list of variables. Returns the root node of the BDD |
| build_robdd | Optimizes the BDD by reusing previously created nodes through memorization. Returns the optimized BDD |
| evaluate_expression | Evaluates a given Boolean expression with a set of variable assignments and returns the result (0 or 1) |
| draw_bdd | Draws the BDD on a canvas by placing nodes (circles) and edges (lines), recursively positioning them |
| create_networkx_graph | Convert the BDD into a directed graph using NetworkX. Adds nodes and edges with attributes (such as color and label) |

## 4. GUI CODE WALKTHROUGH

This is the main part of the GUI code, and it contains all the Canvas edits and attributes of frames and canvas as well as variables used for building BDD diagram and building ROBDD diagram.

```python
class BDDGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("BDD and ROBDD Visualizer")
        root.configure(bg="gray25")
        self.builder = BDDBuilder()
        self.nx_graph = None
        self.fig = None
        self.toplevel = None

        # Variables for storing expressions and selected orders
        self.expression1_var = tk.StringVar()
        self.expression2_var = tk.StringVar()
        self.selected_order1 = tk.StringVar()
        self.selected_order2 = tk.StringVar()

        # Input for boolean expressions
        tk.Label(root, text="Boolean Expression 1 (e.g., A & B | ~C):", bg="black", fg="white").pack()
        self.expression1_entry = tk.Entry(root, width=50, textvariable=self.expression1_var)
        self.expression1_entry.pack()

        tk.Label(root, text="Boolean Expression 2 (e.g., A & B | ~C):", bg="black", fg="white").pack()
        self.expression2_entry = tk.Entry(root, width=50, textvariable=self.expression2_var)
        self.expression2_entry.pack()

        # Frame for dropdown 1
        dropdown1_frame = tk.Frame(root, bg="gray25")
        dropdown1_frame.pack(pady=5)

        # Label for the first dropdown list
        tk.Label(dropdown1_frame, text="Variable Order 1:", bg="black", fg="white").pack(side="left", padx=5)

        # Dropdown menu for variable ordering of expression 1
        self.order_options1 = ttk.Combobox(root, textvariable=self.selected_order1, state="readonly", width=47)
        self.order_options1.pack(pady=5)

        # Frame for dropdown 2
        dropdown2_frame = tk.Frame(root, bg="gray25")
```

**FIGURE 7 GUI #1**

```python
        dropdown2_frame.pack(pady=5)

        # Label for the second dropdown list
        tk.Label(dropdown2_frame, text="Variable Order 2:", bg="black", fg="white").pack(side="left", padx=5)

        # Dropdown menu for variable ordering of expression 2
        self.order_options2 = ttk.Combobox(root, textvariable=self.selected_order2, state="readonly", width=47)
        self.order_options2.pack(pady=5)

        # Bind event to update dropdown when expressions change
        self.expression1_var.trace("w", self.update_dropdowns)
        self.expression2_var.trace("w", self.update_dropdowns)

        # Buttons
        tk.Button(root, text="Build BDDs", bg="black", fg="white", activebackground="#D25A3E", command=self.build_bdds).pack(pady=5)
        tk.Button(root, text="Check Equivalence and Build ROBDD", bg="black", fg="white", activebackground="#D25A3E", command=self.check_equivalence).pack(pady=5)
        tk.Button(root, text="Quit", bg="maroon", fg="white", command=root.quit).pack(pady=5)

        # Frames and canvases for BDDs
        self.frame1 = tk.Frame(root)
        self.frame1.pack(side="left", padx=10)
        self.frame2 = tk.Frame(root)
        self.frame2.pack(side="right", padx=10)

        self.canvas1 = tk.Canvas(self.frame1, width=750, height=700, bg="gray", highlightbackground="gray24")
        self.canvas1.pack()
        self.canvas2 = tk.Canvas(self.frame2, width=750, height=700, bg="gray", highlightbackground="gray24")
        self.canvas2.pack()

        # Frame and canvas for ROBDD
        self.robdd_frame = tk.Frame(root)
        self.robdd_frame.pack(side="left", padx=10)
        self.robdd_canvas = tk.Canvas(self.robdd_frame, width=800, height=800, bg="black")
```

**FIGURE 8 GUI #2**

```python
        self.robdd_canvas.pack()

        self.bdd1_root = None
        self.bdd2_root = None
        self.robdd1_root = None
        self.robdd2_root = None

    def extract_variables(self, expression):
        expression = self.normalize_expression(expression)
        return sorted(set(re.findall(r'\b[a-zA-Z_]\w*\b', expression)) - {"and", "or", "not", "True", "False"})

    def normalize_expression(self, expression):
        expression = expression.replace("~", "not ").replace("&", " and ").replace("|", " or ")
        return expression

    def update_dropdowns(self, *args):
        expression1 = self.expression1_entry.get().upper()
        expression2 = self.expression2_entry.get().upper()

        variables1 = self.extract_variables(expression1)
        variables2 = self.extract_variables(expression2)

        self.update_order_options(self.order_options1, variables1)
        self.update_order_options(self.order_options2, variables2)

    def update_order_options(self, combobox, variables):
        if variables:
            orders = [" ".join(perm) for perm in permutations(variables)]
            combobox['values'] = orders
            combobox.current(0)  # Select the first permutation by default
        else:
            combobox['values'] = []
            combobox.set('')

    def build_bdds(self):
        expression1 = self.expression1_entry.get().upper()
```

**FIGURE 9 GUI #3**

```python
        expression2 = self.expression2_entry.get().upper()
        selected_order1 = self.selected_order1.get().split()
        selected_order2 = self.selected_order2.get().split()

        if not expression1 or not expression2:
            messagebox.showerror("Error", "Please enter two Boolean expressions.")
            return

        normalized_expression1 = self.normalize_expression(expression1)
        normalized_expression2 = self.normalize_expression(expression2)

        variables1 = self.extract_variables(normalized_expression1)
        variables2 = self.extract_variables(normalized_expression2)

        if not variables1 or not variables2:
            messagebox.showerror("Error", "No variables found in one or both of the expressions.")
            return

        # Use the selected order for building BDDs
        variables1 = selected_order1 if selected_order1 else variables1
        variables2 = selected_order2 if selected_order2 else variables2

        try:
            # Build the BDDs with the selected variable order
            self.bdd1_root = self.builder.build_bdd(normalized_expression1, variables1)
            self.bdd2_root = self.builder.build_bdd(normalized_expression2, variables2)

            # Clear existing drawings
            self.canvas1.delete("all")
            self.canvas2.delete("all")
            self.robdd_canvas.delete("all")

            # Draw the BDDs
            self.builder.draw_bdd(self.canvas1, self.bdd1_root, 400, 50, dx=150)
            self.builder.draw_bdd(self.canvas2, self.bdd2_root, 400, 50, dx=150)
```

**FIGURE 10 GUI #4**

```python
            messagebox.showinfo("Success", "BDDs Built and Displayed with selected variable order.")
        except ValueError as e:
            messagebox.showerror("Error", str(e))

    def check_equivalence(self):
        if not self.bdd1_root or not self.bdd2_root:
            messagebox.showerror("Error", "Please build the BDDs first.")
            return

        try:
            # Reset the builder's cache and build ROBDDs
            self.builder.node_cache = {}
            self.robdd1_root = self.builder.build_robdd(self.bdd1_root)
            self.builder.node_cache = {}
            self.robdd2_root = self.builder.build_robdd(self.bdd2_root)

            # Convert ROBDDs to NetworkX graphs
            graph1 = self.builder.create_networkx_graph(self.robdd1_root, 1)
            graph2 = self.builder.create_networkx_graph(self.robdd2_root, 2)

            # Compare equivalence
            def compare_nodes(node1, node2):
                if node1 is None and node2 is None:
                    return True
                if node1 is None or node2 is None:
                    return False
                if node1.is_terminal() and node2.is_terminal():
                    return node1.value == node2.value
                return compare_nodes(node1.low, node2.low) and compare_nodes(node1.high, node2.high)

            if compare_nodes(self.robdd1_root, self.robdd2_root):
                messagebox.showinfo("Equivalence Check", "The two boolean functions are equivalent.")
            else:
                messagebox.showinfo("Equivalence Check", "The two boolean functions are not equivalent.")
```

**FIGURE 11 GUI #5**

```
            # Visualize the graphs
            self.show_matplotlib_graph(graph1, graph2)

        except Exception as e:
            messagebox.showerror("Error", f"Failed to build ROBDDs or check equivalence: {e}")

    def show_matplotlib_graph(self, nx_graph1, nx_graph2, graph_type="Default_Type"):
        try:
            pos1 = nx.multipartite_layout(nx_graph1, subset_key="subset_key")
            pos2 = nx.multipartite_layout(nx_graph2, subset_key="subset_key")
        except Exception as e:
            print(f"Layout Error: {e}")
            pos1 = nx.spring_layout(nx_graph1)
            pos2 = nx.spring_layout(nx_graph2)

        if self.fig:
            plt.close(self.fig)

        self.fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 7))
        self.fig.canvas.manager.set_window_title("ROBDD Graphs Display")

        # Extract node attributes for labels and ensure they are correct
        labels1 = {node: data['label'] for node, data in nx_graph1.nodes(data=True)}
        labels2 = {node: data['label'] for node, data in nx_graph2.nodes(data=True)}

        # Draw Graph 1
        node_colors1 = [data['color'] for _, data in nx_graph1.nodes(data=True)]
        edge_colors1 = [data['color'] for _, _, data in nx_graph1.edges(data=True)]

        nx.draw(
            nx_graph1,
            pos=pos1,
            with_labels=True,
            ax=ax1,
            labels=labels1,  # Ensure the correct labels are passed
            node_color=node_colors1,
```

FIGURE 12 GUI #6

```python
            edge_color=edge_colors1,
            edge_cmap=plt.cm.Reds
        )
        ax1.set_title(f"ROBDD of Boolean expression 1")

        # Draw Graph 2
        node_colors2 = [data['color'] for _, data in nx_graph2.nodes(data=True)]
        edge_colors2 = [data['color'] for _, _, data in nx_graph2.edges(data=True)]

        nx.draw(
            nx_graph2,
            pos=pos2,
            with_labels=True,
            ax=ax2,
            labels=labels2,  # Ensure the correct labels are passed
            node_color=node_colors2,
            edge_color=edge_colors2,
            edge_cmap=plt.cm.Greens
        )
        ax2.set_title(f"ROBDD of Boolean expression 2")

        plt.show()
        canvas = FigureCanvasTkAgg(self.fig, master=self.robdd_canvas)
        canvas_widget = canvas.get_tk_widget()
        canvas_widget.pack()
        canvas.draw()
        plt.show(block=False)  # Kept to avoid problems with canvas refresh

if __name__ == "__main__":
    root = tk.Tk()
    app = BDDGUI(root)
    root.mainloop()
```

**FIGURE 13 GUI #7**

# A. Methods Break down

## "Constructor Method (__init__)":

The BDDGUI class defines a graphical user interface (GUI) for visualizing Binary Decision Diagrams (BDDs) and Reduced Ordered Binary Decision Diagrams (ROBDDs). This is achieved using the tkinter library to create the interface, along with integration of the BDDBuilder class to handle the logic for building and drawing BDDs and ROBDDs.

### Attributes:

- **root:** The main window of the Tkinter GUI.
- **builder**: An instance of the BDDBuilder class used for creating BDDs and ROBDDs.
- **nx_graph**: Placeholder for NetworkX graph.
- **fig**: Placeholder for the matplotlib figure.
- **toplevel:** Placeholder for top-level window.
- **expression1_entry, expression2_entry**: Text entry fields for Boolean expressions.
- **variable_order_entry:** Text entry for variable order.
- **canvas1, canvas2**: Canvases for drawing the two BDDs.
- **robdd_canvas**: Canvas for drawing the ROBDD.
- **bdd1_root, bdd2_root**: Root nodes of the two BDDs.
- **robdd1_root, robdd2_root**: Root nodes of the two ROBDDs.

## “Extract_variables”:

This method takes a Boolean expression as input, normalizes it, and extracts all variables (alphabetic characters) used in the expression.

**Parameters:**

- **expression:** Boolean expression to extract variables from.

**Returns:**

- List of sorted variables found in the expression, excluding reserved keywords.

## “Normalize_expression”:

This method standardizes the given Boolean expression by replacing symbols like ~, &, and | with Python-compatible equivalents (not, and, or).

**Parameters:**

- **expression**: The Boolean expression to normalize.

**Returns:**

- Normalized Boolean expression with standardized operators.

## “Validate_and_order_variables”:

This method validates the user-defined variable order and reorders the variables based on that input. If no order is provided, the method defaults to sorting the variables alphabetically.

**Parameters:**

- **variables:** List of extracted variables from the Boolean expressions.
- **user_order**: A comma-separated string representing the user-defined variable order.

**Returns:**

- A list of variables arranged according to the user-defined or alphabetically sorted order.

## “Build_bdds”:

This method builds the Binary Decision Diagrams (BDDs) for the two Boolean expressions entered by the user. It also draws the resulting BDDs on two separate canvases within the GUI.

**Parameters:**

- None (uses the entries in the GUI).

**Process:**

- Retrieves the Boolean expressions from the entry fields.
- Normalizes and extracts the variables.
- Validates and reorders the variables.
- Build BDDs using the BDDBuilder class.
- Draws the BDDs on the respective canvases.
- Displays a success or error message based on the operation result.

## “Check_equivalence”:

This method checks if the two Boolean expressions are equivalent by comparing their corresponding Reduced Ordered Binary Decision Diagrams (ROBDDs). It also visualizes the ROBDDs.

**Parameters:**

- None (uses the BDDs built previously).

**Process:**

- Ensures that both BDDs are built.
- Builds the ROBDDs from the BDDs using the BDDBuilder class.
- Compare the ROBDDs for equivalence by checking node values recursively.
- Displays a message indicating whether the Boolean expressions are equivalent.
- Visualizes the ROBDDs using matplotlib and NetworkX for graph representation.

## “Show_matplotlib_graph”:

This method visualizes the ROBDDs of the two Boolean expressions using the NetworkX library and matplotlib.

**Parameters:**

- **nx_graph1:** NetworkX graph for the first ROBDD.
- **nx_graph2:** NetworkX graph for the second ROBDD.
- **graph_type:** Optional parameter to specify the type of graph.

**Process:**

- Positions the nodes of both graphs using the multipartite layout for hierarchical visualization.
- Draws the two graphs side-by-side using matplotlib.
- Displays node and edge labels, coloring based on node attributes.
- Embed the matplotlib figure into the Tkinter canvas for display.

## “GUI Layout and Interaction”:

**Boolean Expression Inputs:**

- The GUI provides two input fields for entering Boolean expressions.
- Users can specify a custom order for the variables in a comma-separated format.

**Buttons:**

- **Build BDDs**: Builds and visualizes the BDDs of the two expressions.
- **Check Equivalence and Build ROBDD:** Builds the ROBDDs and checks the equivalence of the two expressions.
- **Quit**: Closes the application.

**Canvas and Frames:**

- The interface has two main sections for displaying the BDDs and one for displaying the ROBDD.
- Each BDD is drawn on a separate canvas, while the ROBDD is drawn on its own dedicated canvas.

## “Error Handling”:

The application provides **error handling** for cases such as:

- Missing or invalid Boolean expressions.
- No variables found in the expressions.
- Failure to build the BDDs or ROBDDs.
- Inability to compare equivalence if BDDs haven't been built.

## “Dependencies”:

**Libraries:**

- tkinter: For creating the GUI.
- re: For regular expression-based variable extraction.
- matplotlib: For graph visualization.
- networkx: For handling and visualizing the ROBDDs as graphs.

## “Example Usage”:

**Building BDDs:**

- The user enters two Boolean expressions (e.g., A & B | ~C).
- The user can optionally specify the order of variables (e.g., A, B, C).
- After clicking "Build BDDs", the program generates the BDDs for the two expressions and displays them on separate canvases.

**Checking Equivalence:**

- After building the BDDs, the user can click "Check Equivalence and Build ROBDD".
- The program compares the two Boolean expressions by building their corresponding ROBDDs and checking if they are equivalent.
- The ROBDDs are visualized side-by-side, and a message is shown indicating whether the expressions are equivalent.

## B. Table of Methods

| Method | Description |
|---|---|
| **__init__(self, root)** | **Initializes the GUI components, including labels, entry fields, buttons, and canvases** |
| **extract_variables(self, expression)** | **Extracts and returns a sorted list of unique variables from a Boolean expression** |
| **normalize_expression(self, expression)** | **Normalizes the Boolean expression by replacing operators** |
| **validate_and_order_variables(self, variables, user_order)** | **Validates and orders the list of variables based on user input, and appends remaining variables alphabetically** |
| **build_bdds(self)** | **Builds Binary Decision Diagrams (BDDs) for two Boolean expressions and displays them** |
| **check_equivalence(self)** | **Checks if the two BDDs are equivalent, builds ROBDDs, and compares them** |
| **show_matplotlib_graph(self, nx_graph1, nx_graph2, graph_type="Default_Type")** | **Displays ROBDDs of two Boolean expressions as NetworkX graphs using Matplotlib** |

# 5.GUI VISUALIZATION

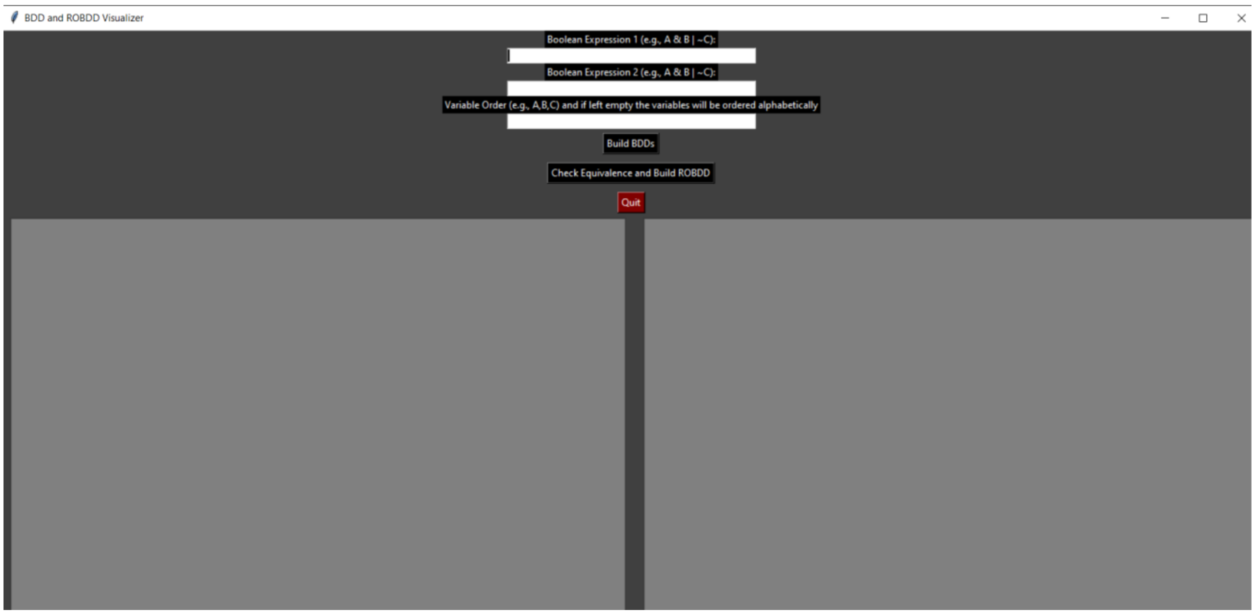



FIGURE 14 GUI GENERAL VIEW

**This Is the GUI general view where you can insert and interact with the available text boxes and produce the BDD and ROBDD .**

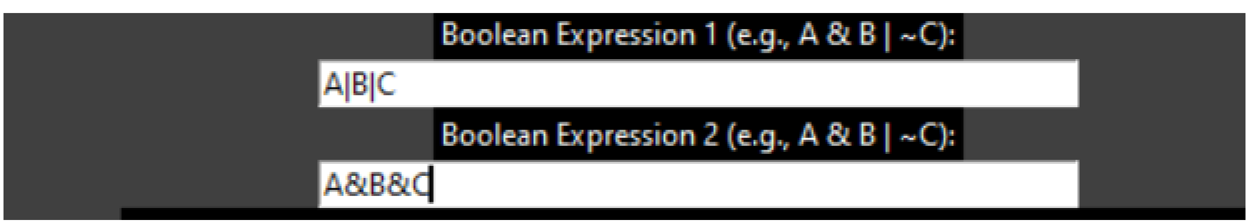



FIGURE 15 GUI INSERT FN TAB

**Here is the text boxes for inserting the expression.**

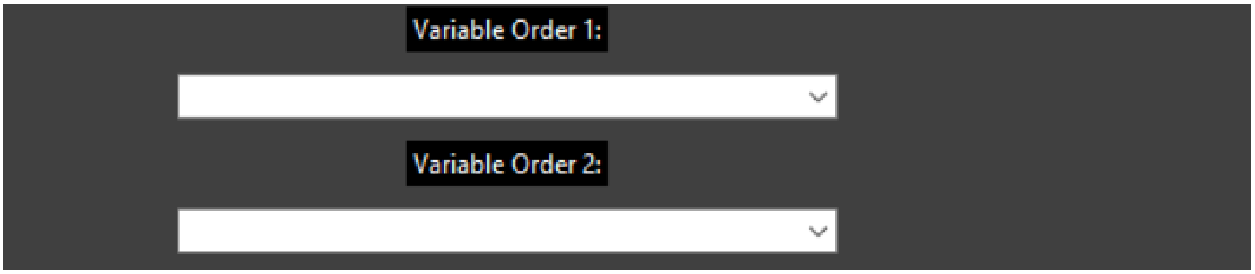



FIGURE 16 GUI VARIABLE INSERT ORDER

**Here we insert the order that we want the variables to be identified with, this can be left blank and have the system use the general alphabetical order.**

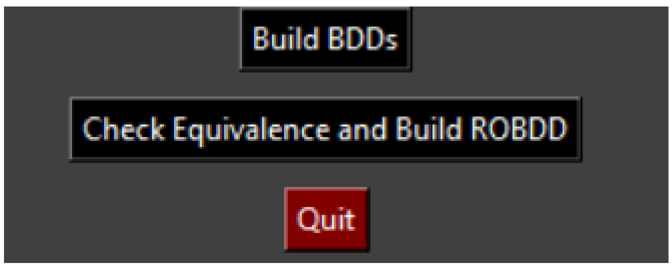



FIGURE 17 GUI INTERACTION BUTTONS

**These are our systems interaction buttons, they do the same functions as written on them.**

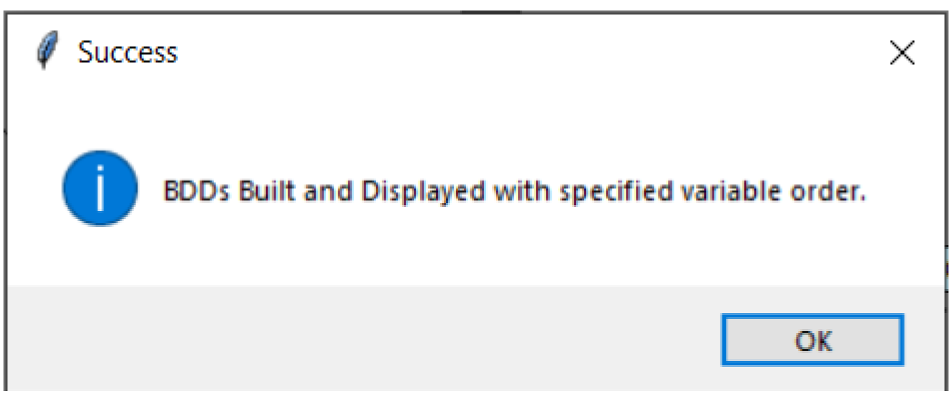



FIGURE 18 GUI SUCCESS CASE

**In case the BDD is successfully built and the inserted functions were correct, this message will be displayed.**

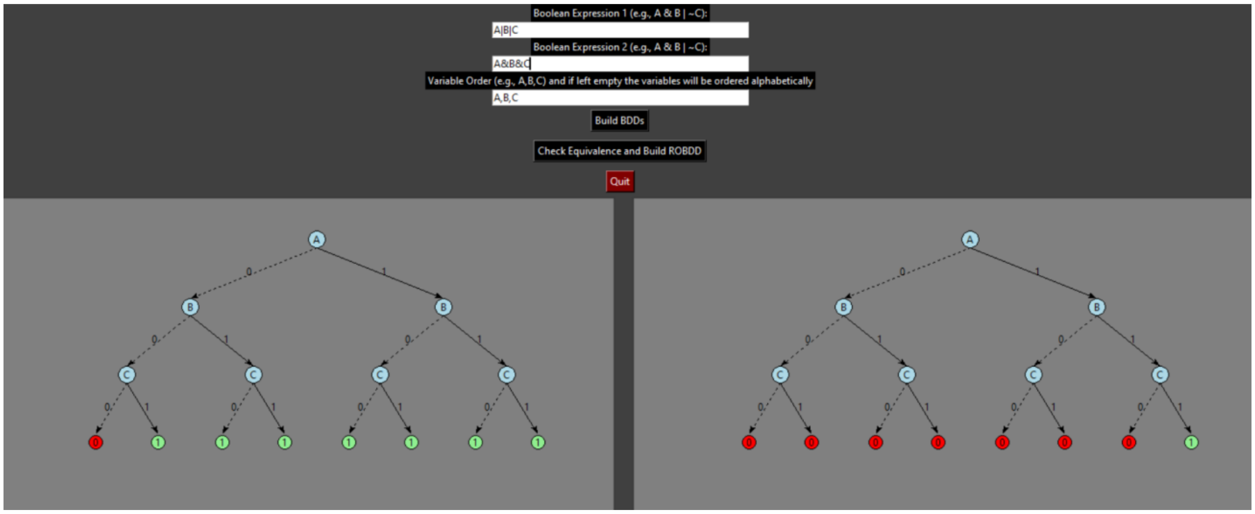



**FIGURE 19 GUI VISIUAL BDD IMPLEMENTATION**

**- The system will provide the BDD’s, with the first on the left and second on the right respectively.**
**- The Variables are given a light blue color .**
**- In case True(1); a green color will be visible.**
**- In case False(0); a red color will be visible.**
**- The dashed lines are an indication to (0) valued lines**

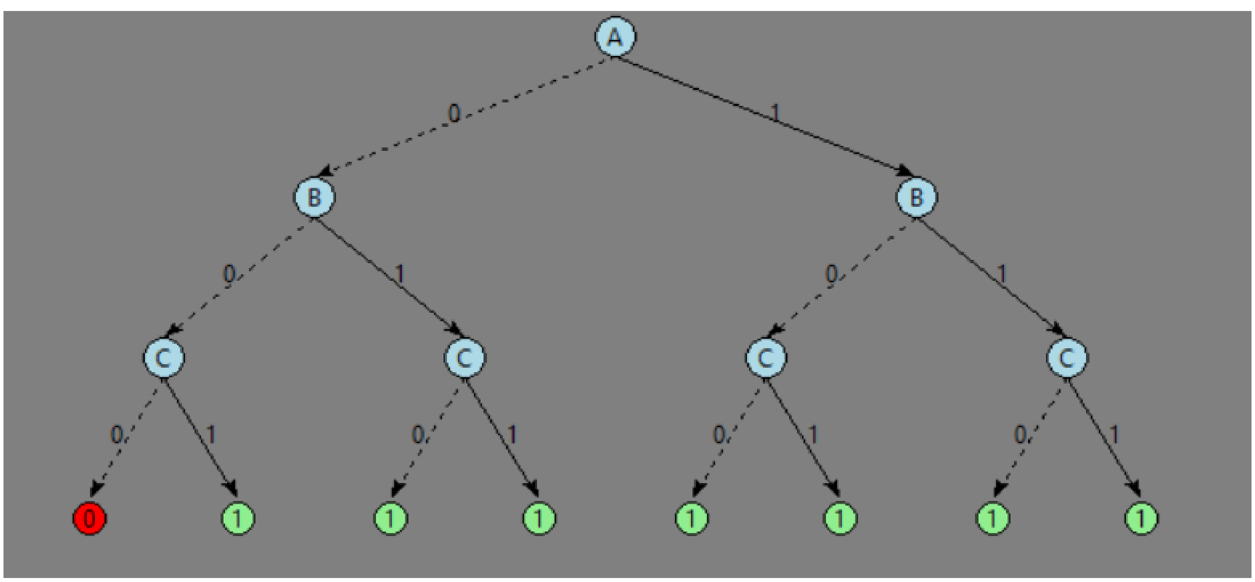



**FIGURE 20 GUI BDD CLOSER LOOK**

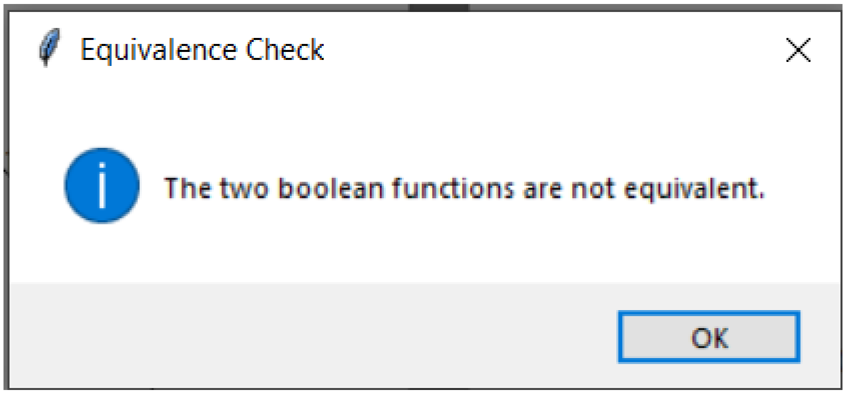



**FIGURE 21 GUI NON-EQUIVALENCE**

**By clicking the " Check Equivalence and Build ROBDD" button we will receive this message in case not equivalent.**

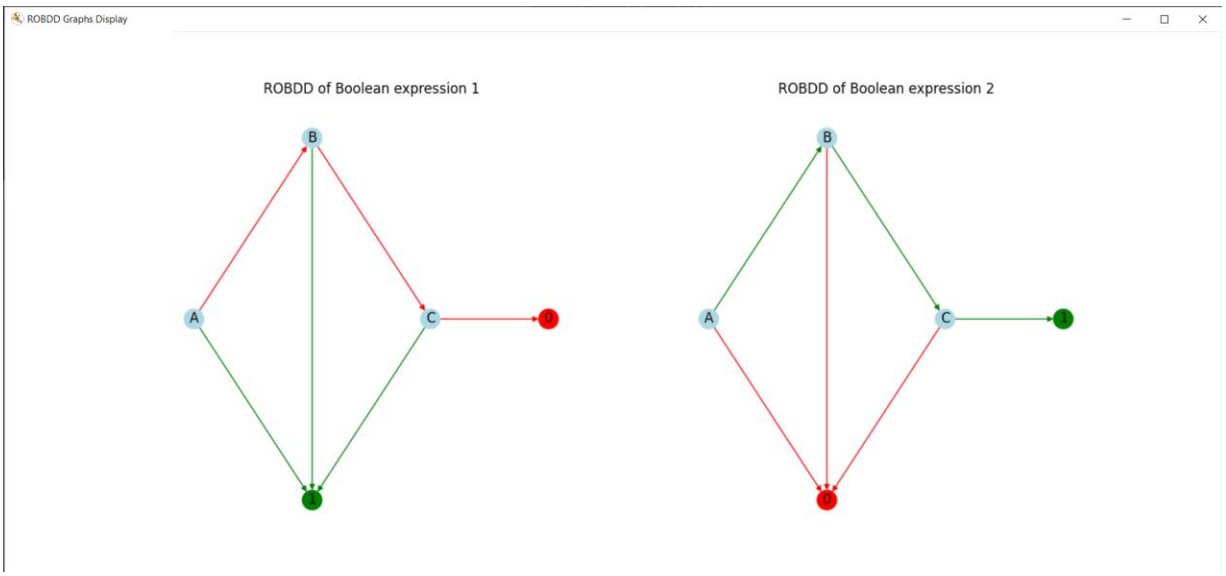
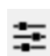



**FIGURE 22 GUI ROBDD**

**- This is the Image of the ROBDD displayed.**
**- The ROBDD will display even if the functions are non-equivalent.**

FIGURE 23 GUI ROBDD PREFRENCES MENU

**These are the interactions of the ROBDD displayed, where each of them has a unique function;**
**- Home button: returns home**
**- Left button: goes to previous view**
**- Right button goes to next view**
**-**  **: fixes aspect and axis**
**- Search button: zooms**
**- Save button: saves ROBDD produced**

**By clicking The configure subplots button** 

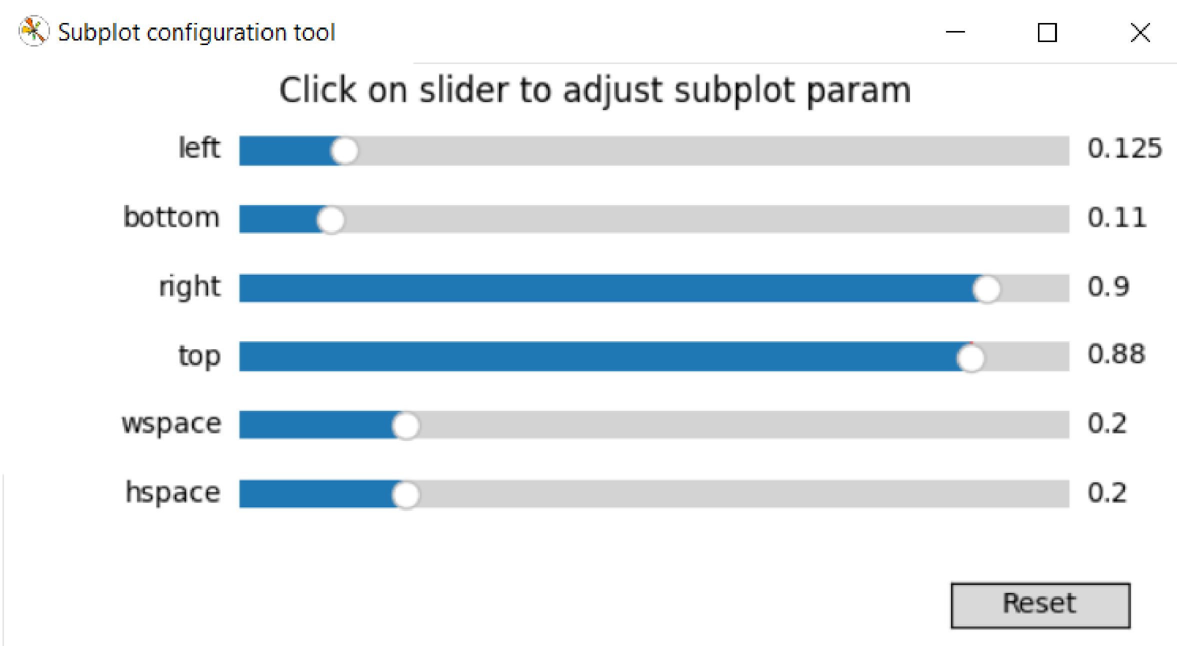



FIGURE 24 GUI ROBDD SUBPLOT

**Brings out a menu of selections and options tab with a drag bar and reset button.**

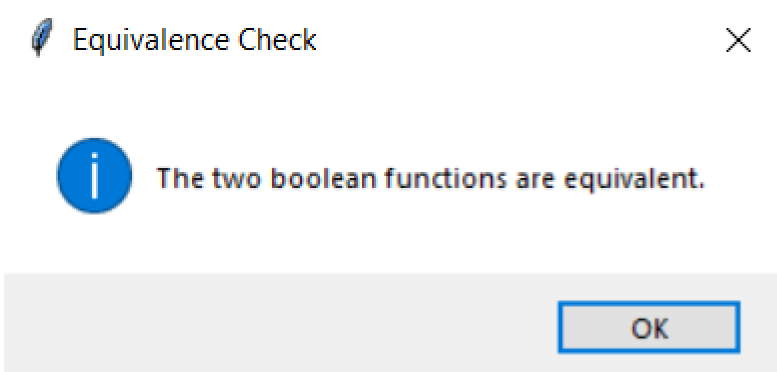



FIGURE 25 GUI EQUIVALENCE

**In case of Equivalence**

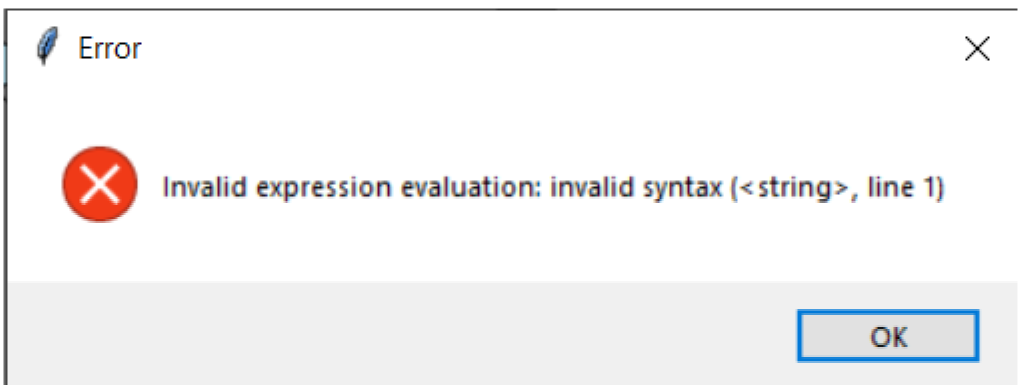



FIGURE 26 GUI ERROR MESSAGE

**In case of Wrong input**

## 6.REFERENCES

**Binary Decision Diagrams (BDDs)**:

- Bryant, R. E. (1986). "Graph-based algorithms for Boolean function manipulation." *IEEE Transactions on Computers*, 35(8), 677–691.

**Reduced Ordered Binary Decision Diagrams (ROBDDs)**:

- Bryant, R. E. (1992). "Symbolic Boolean manipulation with ordered binary decision diagrams." *ACM Computing Surveys (CSUR)*, 24(3), 293-318.

**Boolean Expression Simplification**:

- Huth, M., & Ryan, D. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press.

**Graph Theory and NetworkX**:

- Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). "NetworkX: Network analysis with Python." *Proceedings of the 7th Python in Science Conference*, 11-15.

**Matplotlib and Graph Visualization**:

- Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment." *Computing in Science & Engineering*, 9(3), 90-95.

**Python Programming for Symbolic Computation**:

- Lutz, M. (2013). *Learning Python*. O'Reilly Media.

**ROBDD and BDD Implementation**:

- Karpinski, M., & Shapiro, E. (1997). "On the use of decision diagrams in system reliability analysis." *IEEE Transactions on Reliability*, 46(1), 28-34.

## 6.SUMMARY

The provided code implements a graphical user interface (GUI) for visualizing and manipulating Binary Decision Diagrams (BDDs) and Reduced Ordered Binary Decision Diagrams (ROBDDs) based on user input Boolean expressions. The main features include:

- **Input Fields**: Users can enter two Boolean expressions and specify the order of variables.
- **BDD Construction**: Upon pressing a button, the GUI generates the Binary Decision Diagrams (BDDs) for each expression.
- **ROBDD Comparison**: The GUI allows the user to check if the two BDDs are equivalent by generating and comparing their ROBDD representations.
- **Graphical Representation**: The BDDs and ROBDDs are visually rendered using the NetworkX library for graph drawing, and Matplotlib is used for displaying the ROBDDs side-by-side in an interactive plot.
- **Error Handling**: The application includes error checks for empty input fields, invalid expressions, and missing variables.

The design uses Tkinter for the GUI and integrates several libraries (such as NetworkX and Matplotlib) for graphical rendering. The underlying logic uses the BDDBuilder class to manage the construction, simplification, and drawing of BDDs and ROBDDs, ensuring the efficient representation of Boolean functions.

The code offers a clear user interface for those working with Boolean logic, decision diagrams, and logic optimization, while also providing a solid foundation for further development in symbolic computation and visualization tools.

***Thank You***