Design Description & Activity Diagram

Tyler Dickson

Propositional Logic Calculator

Development of a program that calculates and displays graphical representation of propositional truth tables and logic circuits based on received user input.

Background:
Connectives are symbols or operators used to combine or manipulate logical propositions (statements) to form more complex statements. Connectives allow you to express relationships between propositions and are fundamental for constructing logical expressions and reasoning about them.

Truth tables play important roles in the visualization and problem solving of logical problems, boolean algebra, determining logical equivalence, and making decisions. Boolean math is the basis behind many activities we perform every day such as entering a sentence into a search engine.

The user will input a sequence of logical connectives. The program will output the connective in the binary representation of a truth table or a logic circuit. Several algorithms communicating with various methods will be written to parse and loop through char arrays, variables, and constants.

Primary Programmatic Approach:
Code is written and compiled via C and C++ libraries. Output for truth table would be outputted in binary. Output for Logic circuits will use a program called Graph

Secondary Programmatic Approach:
Android Studio allows a user to import C++ code for Android Applications. Users can add C and C++ code to Android projects by placing the code into a cpp directory in a project module. When the project is built, this code is compiled into a native library that Gradle packages with the app. Java or Kotlin code can then call functions in my native library through the Java Native Interface (JNI). Essentially, functions are written in C++, but called via code written in Java.

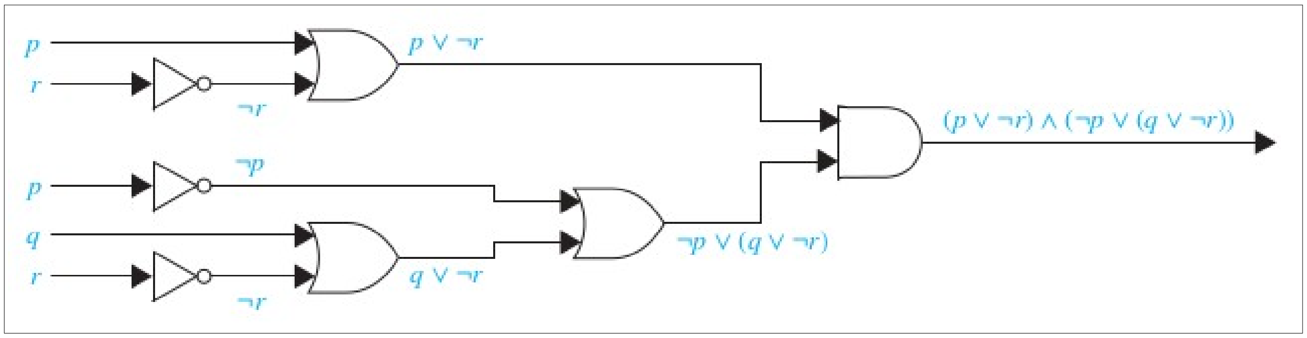
Truth Table Example:
User inputs: $(p \lor \neg r) \land (\neg p \lor (q \lor \neg r))$.

Program outputs:
Program outputs in binary:

```
0  0  0  1
1  0  0  1
0  1  0  1
1  1  0  1
0  0  1  0
1  0  1  0
0  1  1  0
1  1  1  1
```

Logic Circuit Example:
User inputs: (p ∨ ¬r) ∧ (¬p ∨ (q ∨ ¬r)).

Could be displayed as follows:



High Level Programmatic Approach and Design

1. User Input: The program accepts inputs in the form of a logical expression.

    1. The program accepts values for p, q, and r variables. This part of the algorithm needs further refinement to allow for dynamic entry of variables. The diagram represents program output for the inputted value of: (p ∨ ¬r) ∧ (¬p ∨ (q ∨ ¬r))

    2. Logical connectives will be entered as follows: "||" for " ∨ ", "&&" for " ∧ ", "!" for "¬".

    3. Incorrect entries are identified and handled.

2. User Input: The user chooses between Logic circuit and Truth Table Generation.

3. Logic Gate Parser: The program parses the user input – Generating the chosen graphical representation.

4. Truth Table Generation: Converts the logical structure of connectives into a graphical representation of a truth table.

    1. The program will utilize the getBit function which returns the value of a bit within the inputted values of p, q, and r.

    2. We initialize input values as boolean values. This part of the algorithm needs further refinement to allow for dynamic entry of variables.

    3. The program will loop through possible values to determine possible truth table combinations. This is determined by 2^n. For example, two variable inputs represents 4 possible values. Three variable inputs requires 8 possible combinations.

    4. By using the principle of the least significant bit, we can assign values to each input, and dictate the order in which their binary representations are printed. The bitwise ">>" shifts

the bit to the right by whatever we specify (defined in current Algorithm). The &0x1 in line 5 isolates the lease significant bit. If the bit at the current position is 1, the boolean value returns true. 0 returns false.

   5. The program evaluates the binary representation, and outputs the display as shown in figure 2, activity diagram.

5. Logic Circuit Generation: Convert the logical structure of connectives into a graphical representation of logic gates. Using Graphviz will allow for future scalability of the program. However, I am unfamiliar with DOT format, and will require more research. Two methods for implementation will be considered going forward.

   Method 1: Graphviz

   1. The program takes the binary representation calculated during the truth table generation and converts into DOT format.

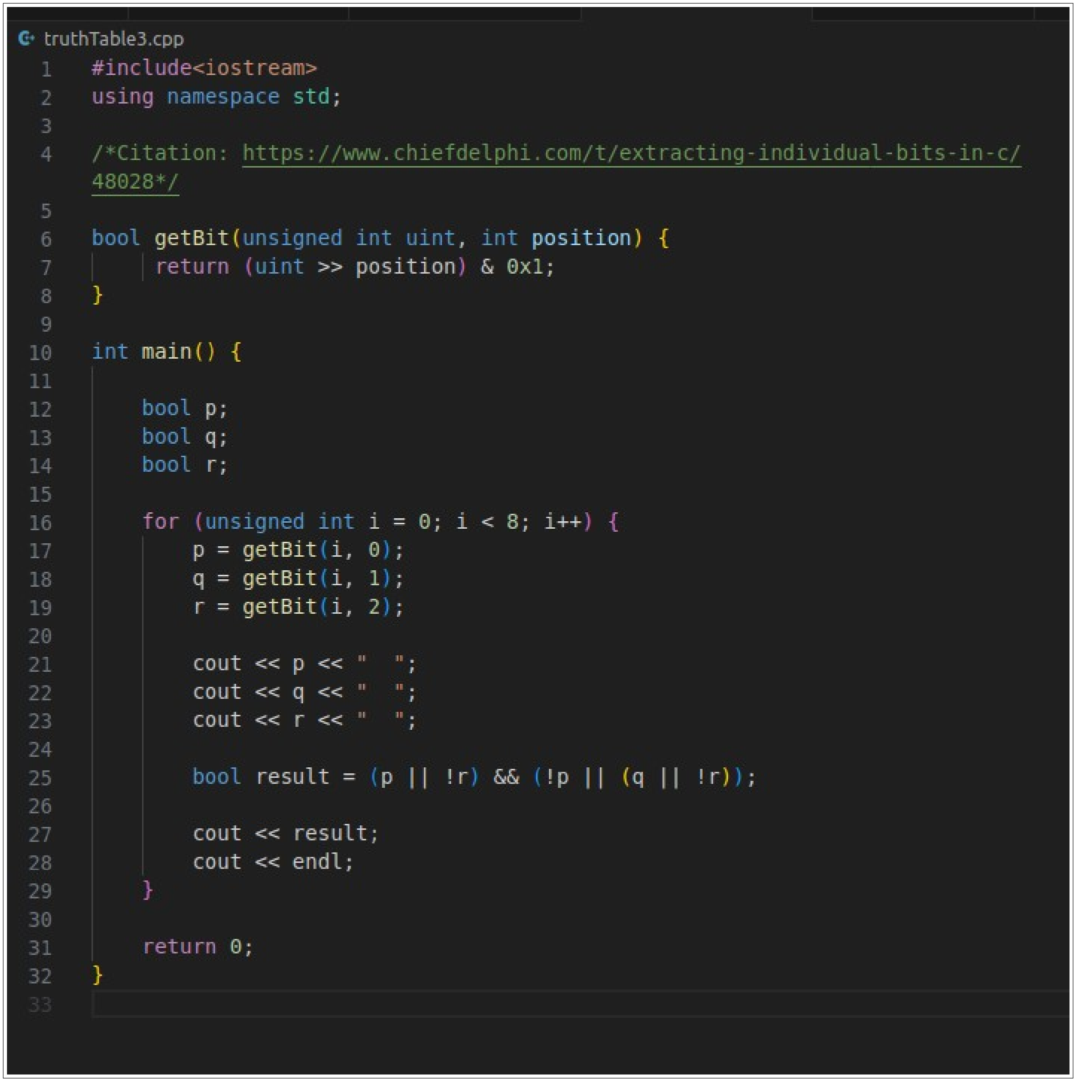
   2. The program defines nodes for every variable and represents the connections between them.

   3. The program uses Graphviz to generate the Image in PNG format.

   Method 2: ASCII art

   1. Pre-defined ASCII art templates are initialized for every logic gate.
   2. The program takes the binary representation calculated during the truth table generation and references a pre-defined ASCII art template to generate the graphical representation.

6. User Input: Allow User to enter a new expression.


truthTable3.cpp

```cpp
#include<iostream>
using namespace std;

/*Citation: https://www.chiefdelphi.com/t/extracting-individual-bits-in-c/
48028*/

bool getBit(unsigned int uint, int position) {
    return (uint >> position) & 0x1;
}

int main() {

    bool p;
    bool q;
    bool r;

    for (unsigned int i = 0; i < 8; i++) {
        p = getBit(i, 0);
        q = getBit(i, 1);
        r = getBit(i, 2);

        cout << p << "  ";
        cout << q << "  ";
        cout << r << "  ";

        bool result = (p || !r) && (!p || (q || !r));

        cout << result;
        cout << endl;
    }

    return 0;
}
```

Figure 1 – Current Algorithm

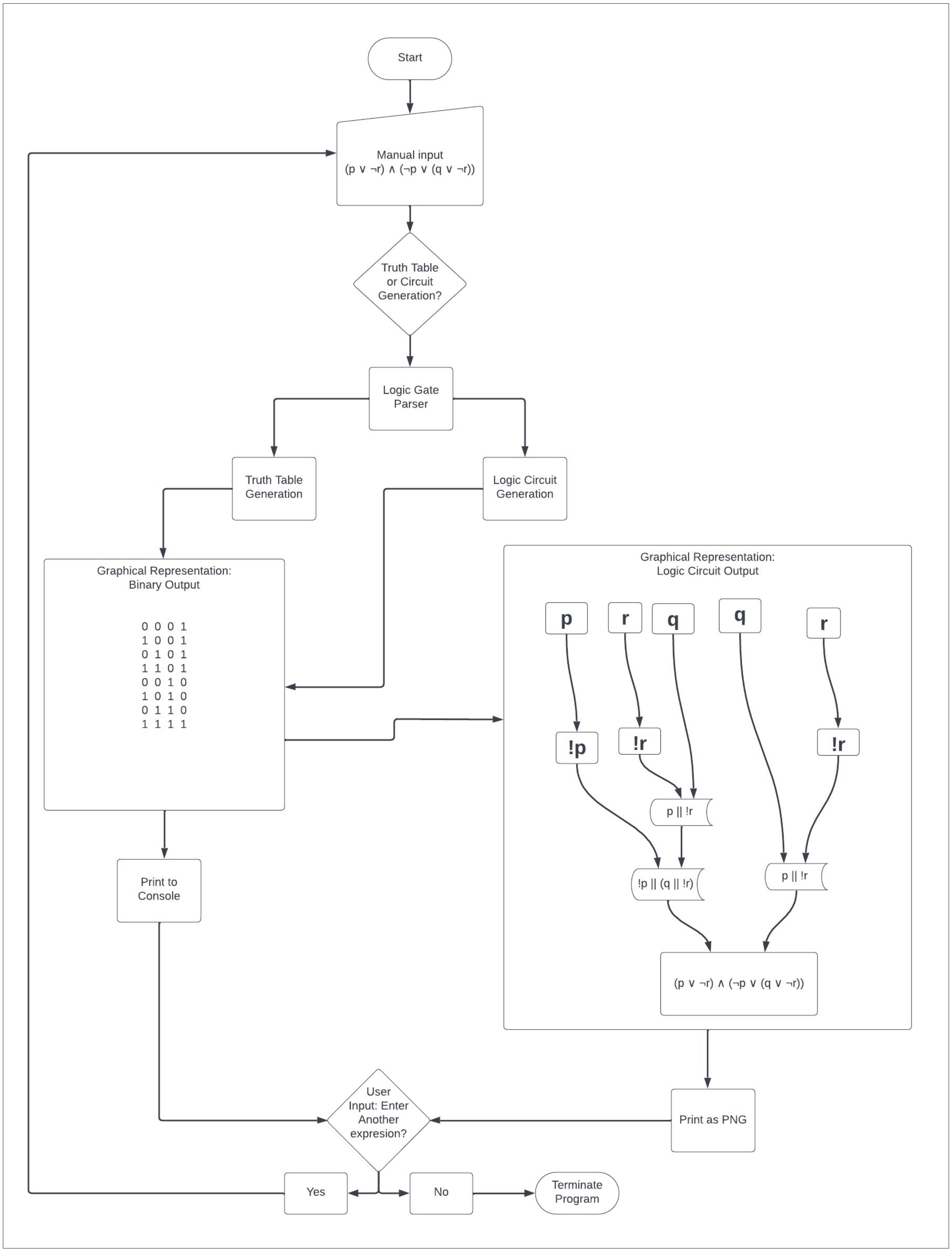


Figure 2 – Activity Diagram

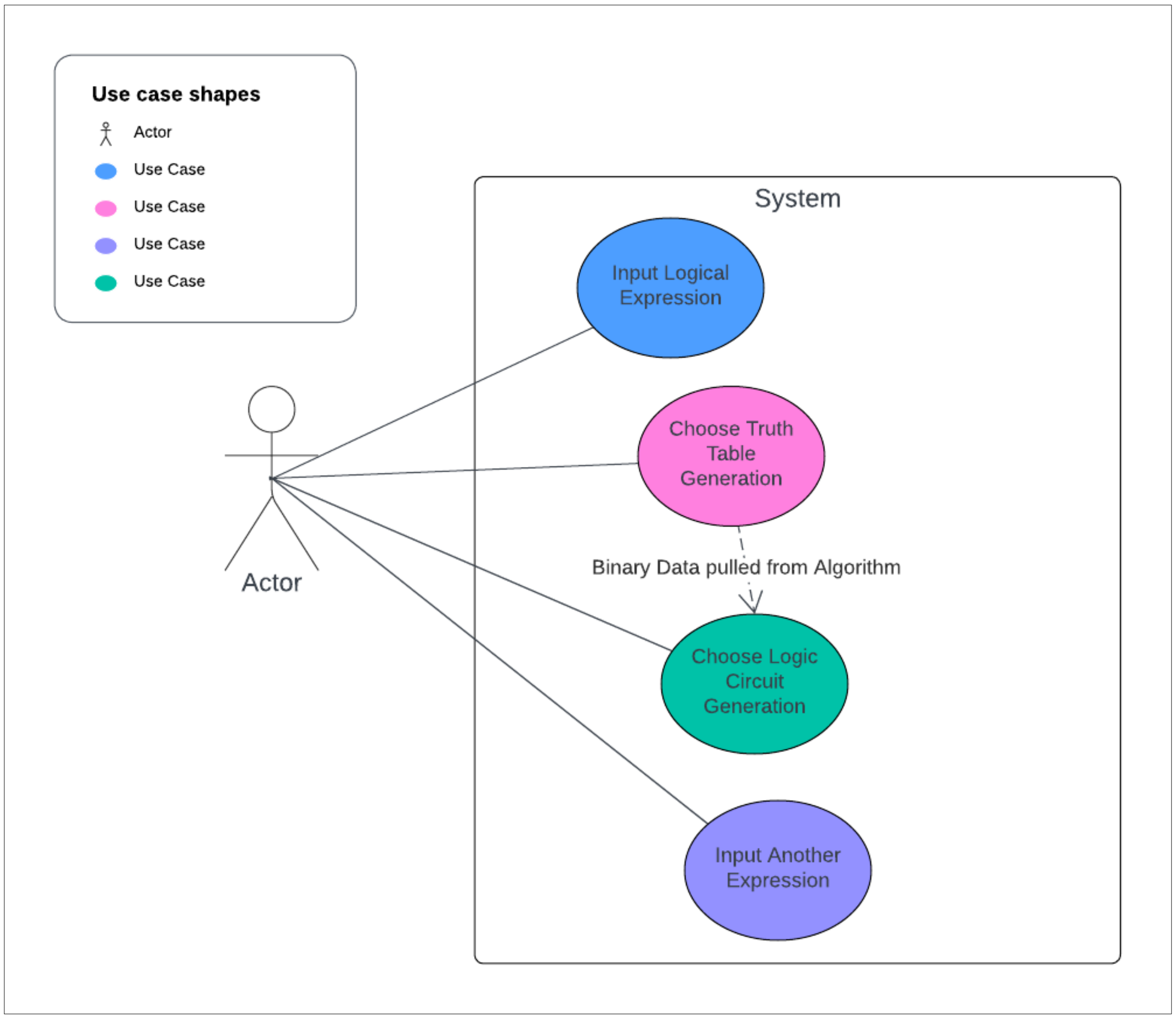


Figure 3 – Use Case Diagram