Fundamental Algorithm Techniques
Problem Set #9

Orysbay Ulykpan

1 Introduction

This report solves Problem Set #9 using simple explanations and clear figures. The goal is not only to present final results, but also to explain the intuition behind them.

The report covers the following topics:

1. counting the number of finite functions that can be implemented on a computer,
2. constructing basic logical operations (NOT, AND, OR) using only NAND gates,
3. explaining why every Boolean function can be computed by a Boolean circuit.

These problems illustrate fundamental ideas from logic, circuits, and theoretical computer science.

2 Problem 1: Finite Functions on a Computer

Finite functions that operate on binary inputs can be written as:

$$F : \{0,1\}^n \to Y.$$

The input set $\{0,1\}^n$ contains exactly 2^n different binary strings. A function F is fully specified once we choose an output value from the set Y for each of these 2^n inputs.

Because the choice of outputs is independent for every input, the total number of different functions is:

$$|Y|^{2^n}.$$

This formula captures the full expressive power of finite functions on a computer.

Required cases

We now consider several important special cases.

- If $Y = \{0,1\}$, then each input is mapped to either 0 or 1. In this case, the total number of Boolean functions is:
$$\#F = 2^{2^n}.$$
- If $Y = \{-1,0,1\}$, then three different outputs are possible for each input, which gives:
$$\#F = 3^{2^n}.$$
- If $Y = \{0,1\}^m$, then each output is an m-bit string. Since $|\{0,1\}^m| = 2^m$, the number of functions is:
$$\#F = (2^m)^{2^n} = 2^{m \cdot 2^n}.$$

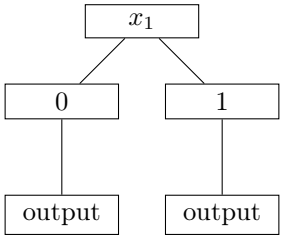

Figure 1: Decision tree: each input path ends at an output.

Decision tree intuition

One intuitive way to understand finite functions is through decision trees. A decision tree queries input bits one by one and branches depending on their values. Eventually, the computation reaches a leaf node that stores the output value.

Each path from the root to a leaf corresponds to exactly one input in $\{0,1\}^n$, and each leaf represents the output chosen for that input.

This visualization makes it clear why the number of functions grows exponentially.

3 Problem 2: NAND $\Rightarrow$ NOT, AND, OR

The NAND gate is one of the most important logical gates. It is defined as:

$$A \uparrow B = \neg(A \wedge B).$$

Despite its simple definition, NAND has remarkable expressive power.

3.1 NOT from NAND

A NOT gate can be constructed by connecting both inputs of a NAND gate to the same variable:

$$\neg A = A \uparrow A.$$

This works because $A \wedge A = A$, and the NAND gate negates this value.

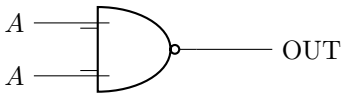

Figure 2: NOT using one NAND gate.

3.2 AND from NAND

An AND gate can be constructed by first applying NAND and then negating the result using another NAND:

$$A \wedge B = (A \uparrow B) \uparrow (A \uparrow B).$$

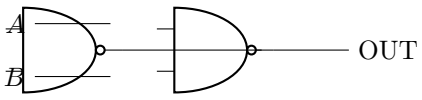

Figure 3: AND using two NAND gates.

3.3 OR from NAND

Using De Morgan's law, an OR gate can be expressed using NAND gates:

$$A \vee B = (A \uparrow A) \uparrow (B \uparrow B).$$

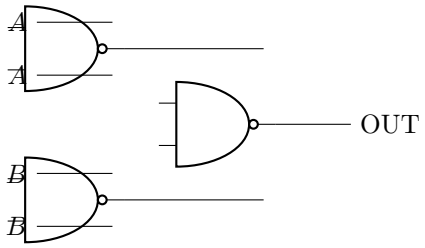

Figure 4: OR using three NAND gates.

Short conclusion

Since all basic logical operations can be implemented using NAND, the NAND gate is called a universal gate.

4 Problem 3: Universality of Boolean Circuits

We now consider Boolean functions of the form:

$$F : \{0,1\}^n \to \{0,1\}.$$

For each input pattern $x \in \{0,1\}^n$, we define an indicator function δ_x that outputs 1 if and only if the input equals x.

Each such block can be implemented using NOT gates (to handle zero bits) and AND gates (to ensure all bits match). The size of each block is $O(n)$.

Any Boolean function can be written as:

$$F(y) = \bigvee_{x:F(x)=1} \delta_x(y).$$

In the worst case, $F(x) = 1$ for all 2^n inputs, which leads to a circuit of size:

$$O(n \cdot 2^n).$$

Structure diagram

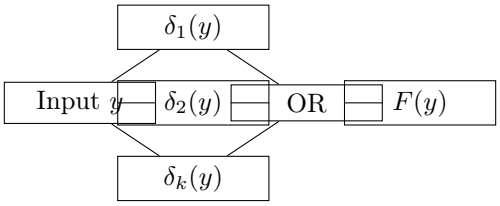

Figure 5: Any Boolean function as OR of matching blocks.

5 Conclusion

- The number of finite functions grows exponentially with the input size.
- The NAND gate alone is sufficient to implement all basic logical operations.
- Any Boolean function can be computed by a circuit, although the size may be exponential in the worst case.