Fundamental Algorithm Techniques
Problem Set #9

Amirtayeva Ainur

1 Introduction

This report solves Problem Set #9 using simple explanations and clear figures. It covers: (1) counting finite functions, (2) building NOT, AND, OR from NAND, (3) explaining why any Boolean function is computable by a circuit.

2 Problem 1: Finite Functions on a Computer

Finite functions can be written as:

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

There are 2^n possible inputs in $\{0,1\}^n$. A function is defined by choosing an output for each input. So the number of different functions is:

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

Required cases

- If $Y = \{0,1\}$: $\#F = 2^{2^n}$.
- If $Y = \{-1,0,1\}$: $\#F = 3^{2^n}$.
- If $Y = \{0,1\}^m$: $\#F = (2^m)^{2^n} = 2^{m \cdot 2^n}$.

Decision tree intuition (simple picture)

A decision tree asks input bits and ends in a leaf storing the output.

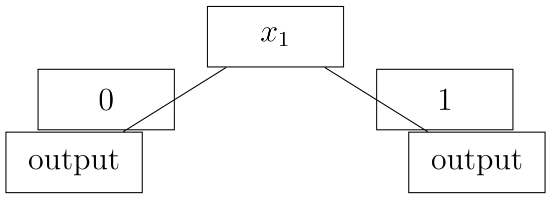

Figure 1: Decision tree idea: each input path ends at an output.

3 Problem 2: NAND $\Rightarrow$ NOT, AND, OR (Clean Schemes)

NAND is defined by:

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

Truth table:

A	B	$A \uparrow B$
0	0	1
0	1	1
1	0	1
1	1	0

Table 1: Truth table of NAND.

3.1 2.1 NOT from NAND

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

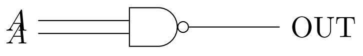

Figure 2: NOT using one NAND (tie inputs).

3.2 2.2 AND from NAND

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

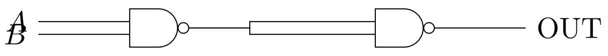

Figure 3: AND using two NAND gates.

3.3 2.3 OR from NAND (Clean and Symmetric)

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

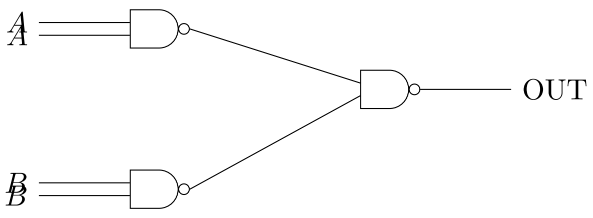

Figure 4: OR using three NAND gates (clean layout like textbook).

Short conclusion

Because we can build NOT, AND, and OR using only NAND, NAND is universal.

4 Problem 3: Universality of Boolean Circuits

We consider:

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

For each input pattern $x \in \{0,1\}^n$, define a small function δ_x that outputs 1 only when the input equals x. This can be done with:

- NOT gates (to match 0-bits),
- AND gates (to require all bits match).

So each δ_x needs $O(n)$ gates.

Then we can write:

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

In the worst case, there are up to 2^n such terms, so the circuit size is:

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

Simple structure diagram

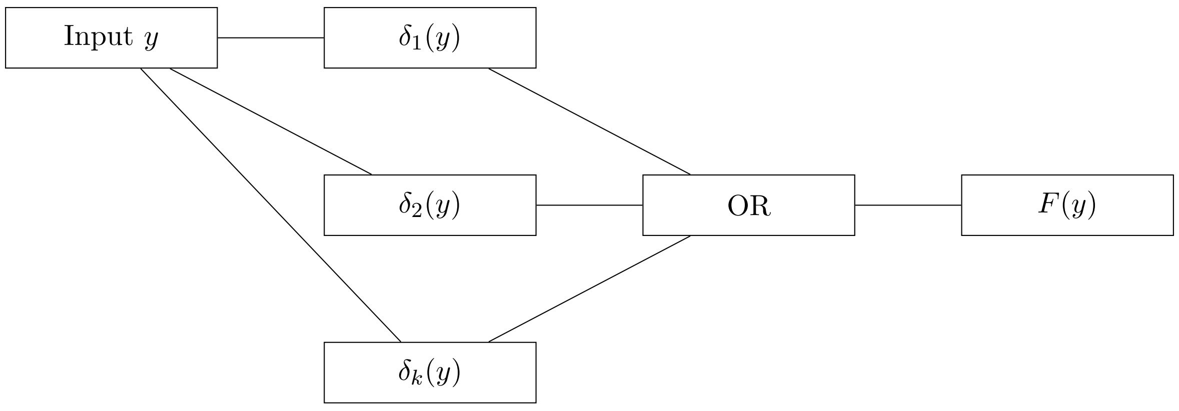

Figure 5: Any Boolean function as OR of matching blocks.

5 Conclusion

- The number of finite functions is $|Y|^{2^n}$ because there are 2^n inputs.
- NAND can generate NOT, AND, and OR, so it is universal.
- Any Boolean function is computable, and a general construction uses $O(n \cdot 2^n)$ gates.