

Quantum Computing

Mathias Niepert

Homework 7

Exercise 1

- 1.1** There are $2^{(2^n)}$ boolean functions $f: \{0,1\}^n \rightarrow \{0,1\}$ because there are 2^n possible input variations and for every variation, there can be 2 possible outcomes of the function.
- 1.2** The crucial question is how many invertible (nonsingular) $n \times n$ matrices with entries $\in \{0,1\}$ exist. A matrix is invertible if and only if its determinant is nonzero.
- 1.2.1** $n=1$: There are 2 boolean maps. M has to be 1 and a can either be 0 or 1.
- 1.2.2** There are 24 boolean maps, because there exist 6 reversible (invertible) matrices and 4 possibilities for \vec{a} .
- $$\det\left(\begin{pmatrix} a & b \\ c & d \end{pmatrix}\right) = ad - cb \neq 0 \text{ for } ad = 1 \text{ and } cb = 0 \text{ (3 possibilities) or } ad = 0 \text{ and } cb = 1 \text{ (3 possibilities).}$$
- 1.2.3** I tried to do it the same way as in 1.2.2 but it turned out that this got very complex. I think there are approximately $178 \cdot 4$.

Exercise 2

- AND: $a \wedge b = \overline{\overline{a \wedge b} \wedge \overline{a \wedge b}}$
 $(AND(a, b) = NAND(NAND(a, b), NAND(a, b)))$
- NOT: $\overline{a \wedge a} = \overline{a} \vee \overline{a} = \overline{a}$
- XOR: $\overline{\overline{\overline{\overline{a \wedge a \wedge b \wedge a \wedge b \wedge b}}}} = \overline{\overline{a \wedge b \wedge a \wedge b}} = \overline{(a \vee \overline{b}) \wedge (\overline{a} \vee b)} = (\overline{a} \wedge b) \vee (a \wedge \overline{b}) \equiv XOR \text{ (disjunctive normal form)}$
 $XOR(a, b) = NAND((NAND(NAND(a, a), b), NAND(a, NAND(b, b))))$

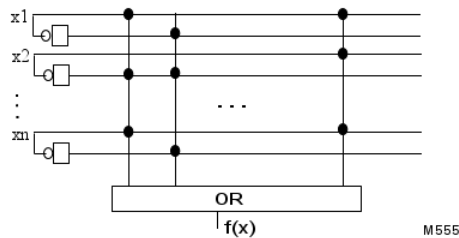


Figure 1: DNF Gate

Exercise 3

One can compute the XOR function in $\{\vee, \wedge, \neg\}$ e.g. using the disjunctive normal form: $XOR(a, b) \equiv (\bar{a} \wedge b) \vee (a \wedge \bar{b})$

One can do this by using e.g. 5 gates (2 NOT, 2 AND and 1 OR). Thus you can implement an half-adder with e.g. 6 gates, one for the carry bit and 5 for the XOR gate. The full-adder uses two half-adder and one AND-gate (carry bit) thus it has 13 gates. To implement an n-bit adder, you need n-1 full-adder and 1 half-adder (Book page 132, Fig. 3.5, 3.6, 3.7). So you need $(n-1)13 + 6 = 13n - 7$ gates. Thus the size of the circuit is in $\text{poly}(n)$.

Exercise 4

The circuit in Figure 1 can compute every boolean function using the disjunctive normal form. The black dots mean that the variables (every of them either negated or not) are fed into an AND gate. The depth of the circuit is ≤ 3 (length of the longest path in the Graph defined by the circuit). Thus any boolean function can be computed by a circuit of depth ≤ 3 with gate set $\{AND, OR, NOT\}$ because every boolean function can be written (get transformed) into the DNF.

Exercise 5

Proof by induction.

Base case: $n = 1$:

We have 4 boolean functions and every of them can be computed by a circuit of size $\leq 5 \cdot 2^{1-1} - 4 = 1$. The identity consists of a single wire, the bit flip uses a single NOT gate, the function which replaces the input with a 0 can be implemented by ANDing the input bit with a bit initially 0 and the function which replaces the input bit with 1 can be implemented by ORing the input bit with a bit initially 1.

Induction step: $n \rightarrow n + 1$

We assume that any functions on n bits can be computed by a circuit and let f be a function of $n+1$ bits. Define n -bit functions f_0 and f_1 by $f_0(x_1, \dots, x_n) \equiv f_0(0, x_1, \dots, x_n)$ and $f_1(x_1, \dots, x_n) \equiv f_1(1, x_1, \dots, x_n)$. We assume that the size of f_0 and f_1 are both smaller than $5 \cdot 2^{n-1} - 4$ (induction hypothesis). We can build a circuit out of these functions like in the book at Figure 3.8 (XOR = OR gate) which computes any f on n bits. The size for this circuit is $\leq 2 \cdot (5 \cdot 2^{n-1} - 4) + 4 = 5 \cdot 2^n - 4$ because we use 2 circuits with size $\leq 5 \cdot 2^{n-1} - 4$ and need 4 additional gates to build the function over n bits.

Exercise 6

$$\text{Let } M = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} M \vec{x} = \begin{pmatrix} x_1 m_{11} \oplus x_2 m_{12} \oplus x_3 m_{13} \\ x_1 m_{21} \oplus x_2 m_{22} \oplus x_3 m_{23} \\ x_1 m_{31} \oplus x_2 m_{32} \oplus x_3 m_{33} \end{pmatrix}$$

m_{11} and m_{22} have to be 1 and m_{12}, m_{13}, m_{22} and m_{23} have to be 0 because the first two bits of the input vector \vec{x} always stay the same. To get the Toffoli function, the third bit has to get flipped ($0 \rightarrow 1/1 \rightarrow 0$), when the first and second are 1 otherwise it stays the same. So we need a function (for the component x'_3 after the circuit) like $x_1 x_2 \oplus x_3$. But this is impossible to do with $x_1 m_{31} \oplus x_2 m_{32} \oplus x_3 m_{33}$ as this is a linear function and $x_1 x_2 \oplus x_3$ is non-linear.