Collaborators : None

Sources :

1) https://en.wikipedia.org/wiki/Disjunctive_normal_form
2) https://mathworld.wolfram.com/DisjunctiveNormalForm.html

Q3 Gates for universal classical computation

3-a)

Disjunctive Normal Form (DNF)

> A DNF formula is a standardized way to represent Boolean functions as a disjunction (OR) of conjunctions (AND) of literals (variables or their negations). Any Boolean function can be expressed in DNF.
> Its basically a **OR of ANDs**, a sum of products.

To show that any Boolean function f:{0,1}^n → {0,1} can be computed using 2-bit AND, 2-bit OR, and NOT gates, we can use the concept of Disjunctive Normal Form (DNF).

Here, each product (conjunction) consists of “n” Boolean inputs where each input is either the initial Boolean input or its negation; and the sum is the disjunction of the “n” conjunctions.

Ex: For 3 boolean inputs p,q and r, an example of DNF function is (p ∧ q ∧ r) V (p ∧ ~q ∧ ~r) V (~p ∧ ~q ∧ r).

Proof : As any Boolean function can be written in DNF form from the Truth table, each disjunction can be considered as a 2-bit OR gate; each conjunction can be solved by a series of 2-bit AND gates and the inputs for the conjunction can be either the original Boolean inputs or their negations which can be outputted from NOT gates.

For example, from the previous example,

(p ∧ q ∧ r) V (p ∧ ~q ∧ ~r) V (~p ∧ ~q ∧ r) => (p AND q AND r) OR (p AND (NOT q) AND (NOT r)) OR ((NOT p) AND (NOT q) AND r)

Proof by Induction : Lets assume we can solve the Boolean function for n-1 inputs with AND, NOT and OR gates.

⇨ $F(a_1, a_2, \dots a_n) = \{ G(a_1, a_2, \dots a_{n-1})\ AND\ a_n \}\ OR\ \{ H(a_1, a_2, \dots a_{n-1})\ AND\ (NOT\ a_n) \}$

In the above equation, G and H can be outputted by AND, NOT and OR gates based on our initial assumption. And we can thus output the n-input F function based on only AND, NOT and OR gates.

3-b)

p OR q ⇔ (p NAND 1) NAND (q NAND 1)

p AND q ⇔ (p NAND q) NAND 1

NOT p ⇔ p NAND 1

Based on the above expression, we can see that AND, OR and Not gates can be represented by only NAND gates. In 3-a we proved that any Boolean function can be represented by AND, OR and NOT gates. Based on these 2 statements, we can say that any Boolean function can be represented by only NAND gates.

Similarly,

p NAND q ⇔ 0 NOR ({a NOR 0} NOR {b NOR 0})

Thus, as NAND can be represented by only NOR gates; and every Boolean function can be represented by only NAND gates, we can infer that every Boolean function can be represented by only NOR gates as well.

3-c)

p XOR 1 = NOT p

p XOR 0 = p (Identity)

p XOR (q XOR 1) = p XNOR q = p XOR (NOT q) = (NOT p) XOR q.

So, whatever operation we do between these states, the output always belong to {0, 1, p, q, NOT p, NOT q, p XOR q, p XNOR q}. We don't have p AND q, p OR q, (NOT p) AND q, (NOT p) AND (NOT q), p AND (NOT q), etc.. within these outputs. So, thus we cant say that a classical Boolean circuit can compute any Boolean function by using the following set of logic gates: 2-bit XOR, and NOT.