Computational Boolean Algebra.

WEEK 1

→ data structures + operators.

Shannon Expansion Theorem

$F(x_1, x_2 \ldots \text{set}(x_i=1), \ldots x_n)$ } Shannon co-factor

$F(x_1, x_2 \ldots \text{set}(x_i=0), \ldots x_n)$ } co-factor

if $x_i=1$ ⇒ positive co-factor.

if $x_i=0$ ⇒ Negative co-factor

* Shannon expansion:

$$F(x_1, x_2, \ldots, x_i \ldots x_n) = x_i \cdot F(x_i=1) + \bar{x}_i \cdot F(x_i=0)$$

mux: pos, neg

Just like a multiplexer.

$F(x,y,z,w) =$

$xy\,F(x=1,y=1) + x\bar{y}\,F(x=1,y=0) + \bar{x}y\,F(x=0,y=1)$

$+ \bar{x}\bar{y}\,F(x=0,y=0)$.

* Co-factor properties - Shannon - cofactor prop.

$F_x \oplus F_{\bar{x}} = (F \oplus F_{\bar{x}})_x$

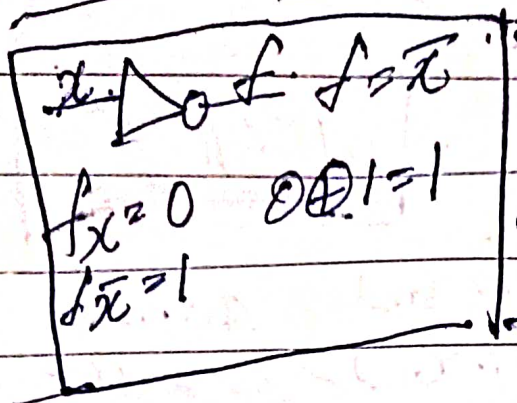
* Boolean derivative: $f_x \oplus f_{\bar{x}}$

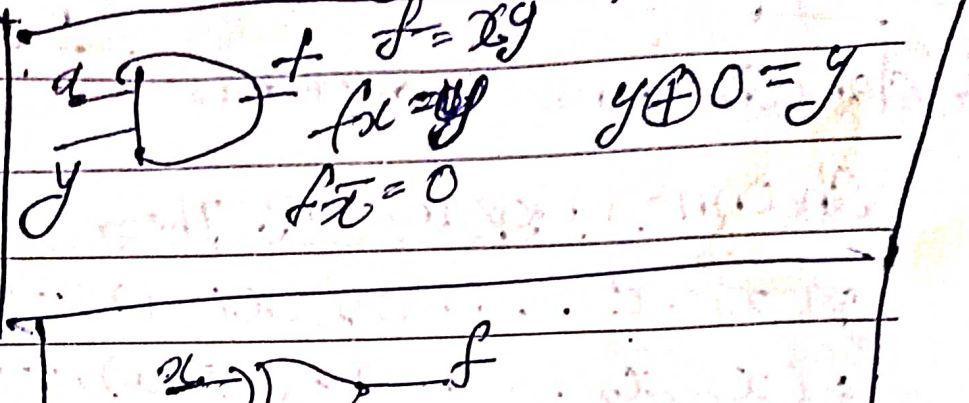
$$\frac{\partial f}{\partial x} = f_x \oplus f_{\bar{x}}$$

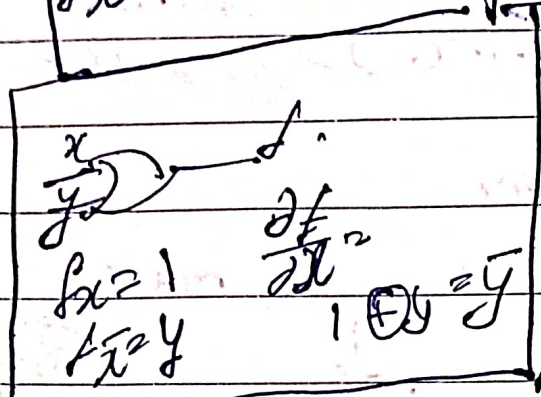
Boolean difference.

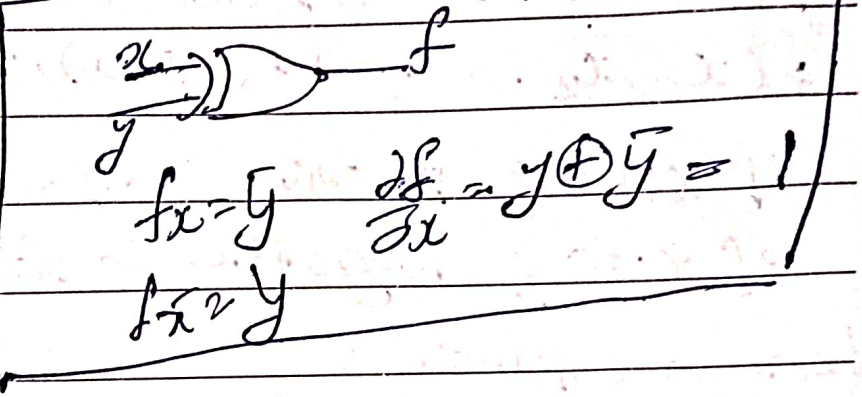
$$\frac{\partial f}{\partial x} = f_x \oplus f_{\bar{x}}$$

Boolean difference for some gates. (2)

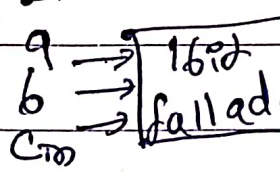








Boolean difference example.



$S = a \oplus b \oplus C_{in}$.

$C_{out} = ab + (a+b)C_{in}$.

$\frac{\partial C_{out}}{\partial C_{in}} = 2$

$\frac{\partial C_{out}}{\partial C_{in}} = ab + (a+b) \oplus ab$.

$= a \oplus b \Rightarrow$ makes sense

$Cout_{Cin} = ab + (a+b)$.

$= a + b$

$Cout_{\bar{C}in} = ab$

when $a \neq b$, output changes

Quantification operators.

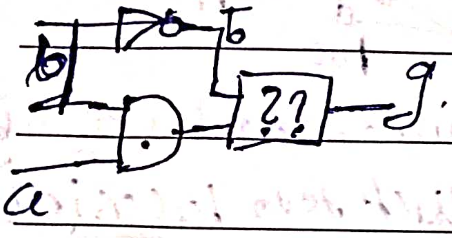
$Fx_i \cdot F\bar{x}_i \Rightarrow$ Universal Quantification $\Rightarrow \forall x F$ for all $F=1$ for all x

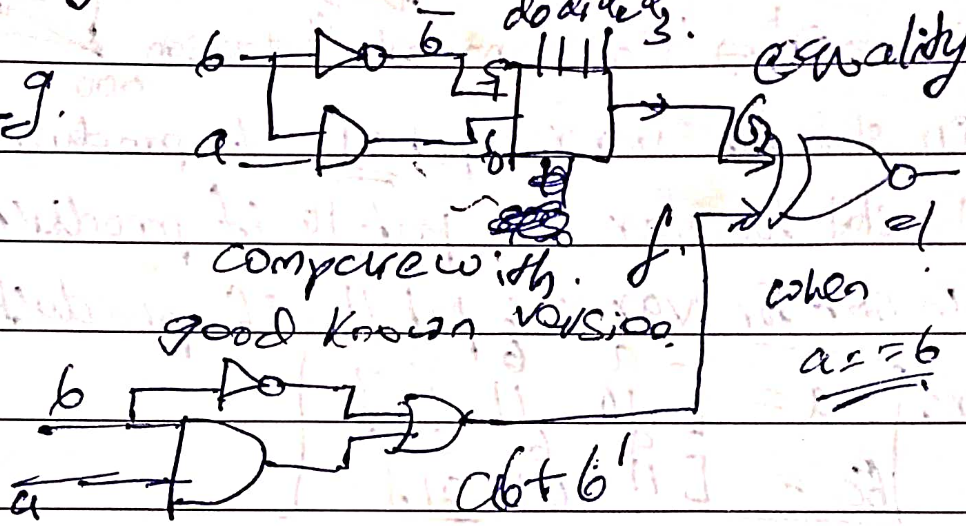
pos neg

$Fx_i + F\bar{x}_i \Rightarrow$ Existential Quantification $\Rightarrow \exists x F$ there exists for an $F=1$ there exists an x

Quantification: Application to Logic Network Repair.

Ex: $ab+b'$ actually $\Rightarrow a+b'$





$G = d_0 s_1' s_0' + d_1 s_1' s_0 + d_2 s_1 s_0' + d_3 s_1 s_0$.

$G = d_0 \bar{a} b \bar{b} + d_1 \bar{a} b \bar{b} + d_2 a b \bar{b} + \sout{d_3 a b b}$

$G = d_0 \bar{a} b + d_1 \bar{b} + d_2 ab$.

$f = ab + b'$

$G \oplus f = G\bar{f} + \bar{G}f \quad (\forall abz) = z_{ab} \cdot z_{ab'} \cdot z_{a'b} \cdot z_{a'b'}$ to be 1.

$z = (d_0 \bar{a} b + d_1 b' + d_2 ab) \oplus (ab + b')$

$z_{ab'} = d_1 \quad z_{a'b} = \bar{d_0} \quad z_{a'b'} = d_1 \quad z_{ab} = d_2$.

$(\forall abz)(d_0 d_1 d_2 d_3) = AND = \bar{d_0} d_1 d_2$ to be 1.

$d_0 = 0$

$d_1 = 1$

$d_2 = 1$

$d_3 = X$

If can be + OR gate, expected

⊕ XOR gate, also repairs.

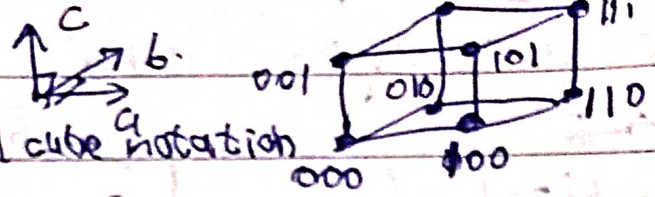
localize the gate

Recursive Tautology

(4)

Set of 'or'ed product terms.

3 var boolean cube



cube = 2^k ⇒ Positional cube notation

In slot for var x_i : put 01 if product term has x in it

In slot for var x', put 10 if product term has $\bar{x}$ in it

In slot for var x_i put 11 if product term has no x or $\bar{x}$

ex.

	a	b	c
$\bar{a}$ =	10	11	11
bc =	11	01	01

$f(a,b,c) = a + bc + ab$ ⇒ [01 11 11], [11 01 01], [01 01 11]

[Result : f is a tautology if and only if f_x & $f_{x'}$ are both tautologies].

That is $f(x=0), f(x=1)$.

$x \cdot 1 + (x=1) + x' \cdot f(x=0) = x \cdot 1 + x' \cdot 1 = x + x' = 1$

URP Implementation (Unique Recursive paradigm)

ex: $f = abd + bc'$

	a	b	c	d	f_a a→1	f_c c→1
abd	01	01	11	01	[11 01 11 01]	[01 01 11 01]
bc'	11	01	10	11	[11 01 10 11]	[11 01 11 11]
						delete

Unate function

ex: $ab + ac'd + c'de' \rightarrow$ unate.

ex: $xy + x'y + xyz' \rightarrow$ not unate.

positive unate & negative unate → binate. (5)

~~a+b'c+ac UNATE~~: ~~a+b'c+bc NOT UNATE~~

a	b	c
[01	11	11]
[11	10	01]
[01	11	01]
U	U	U

a	b	c
[01	11	11]
[11	10	01]
[11	01	01
U	N	U

checking unate.

Unate cube list for f is tautology if it contains all don't care cube [11 ... 11] = 1 → tautology

1st rule: FUN Rules

$f == 1$ ✓

$f_x = 1_2$? [11 11 ... 11] YES

2nd rule: $f == 1$ ✗ NO

01	11	10
11	01	10
01	01	10

is unate but not tautology.

Another rule.

$f_x = 1_2$ YES

x: 01	11	11
x̄: 10	11	11
11	11	10

+ x + x' ... should = 1

Tautology checking - further.

(6)

✓ pick most product terms

✓ pick var with minimum | true var - complement var |

x	y	z	w
01	01	01	01
10	11	01	01
10	11	11	10
01	01	11	01

2-2 = 0, 0, 0, 3-1 = 2

↳ binate 4 cubes | true - compl | = 2-2 = 0.

Algorithm for tautology

```
tautology(f represented in cubelist){
    #check if we can terminate recursion.
    if(f is unate){
        apply unate tautology terminating rules
                directly
        if(==1)
            return(1)
        else
            return(0)
}.
```

else if (any other termination rules, case rules work?) {

return the appropriate if == 1 or == 0

}

else {

// can't tell from this -- find splitting variable

x = most - notunate variable in f.

return (tautology(fx) && tautology(fx'))

}

}

Ex: $f = ab + ac + ab'c' + a'$

	a	b	c
ab	01	01	11
ac	01	11	01
ab'c'	01	10	10
a'	10	11	11
	b	b	b

all are binate

f_a $a \to 1$

[11 01 11]
[11 11 01]
[11 10 10]

$f_{a'}$ $a \to 0$

[11 11 11] ✓ YES.

$f_{a'b} \to 0$

[11 01
11 11] YES, tautology.

YES