

Logic synthesis

RTL \rightarrow gate level netlist

functions:

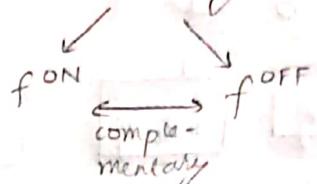
- how to represent

\rightarrow truth table (not practical)

(unique way of representation)

2 funct's can't have same representation

Need to remove redundancy



\rightarrow can evaluate only one

i.e. evaluate minterm or maxterm
unique

{ All these are textual way of
Representing }

\rightarrow Graphical (Binary Decision Diagram)

e.g:

a b c	Z
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

$$Z = f(a, b, c)$$

$$= ab + c$$

Maxterm Representation:

$$Z = (a+b+c) (\bar{a}+\bar{b}+c)$$

$$(\bar{a}+b+c)$$

Shannon's expansion

$$f(x_1, \dots, x_n) = \underbrace{x_i \cdot f(x_1, x_2, \dots, 1, \dots, n)}_{\text{co-factor}} + \underbrace{\bar{x}_i \cdot f(x_1, \dots, 0, \dots, \bar{x}_i)}_{\text{co-factor}}$$

$\Rightarrow f_{x_i}$ $f_{\bar{x}_i}$

$$\Rightarrow \text{if } Z = ab + c = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i}$$

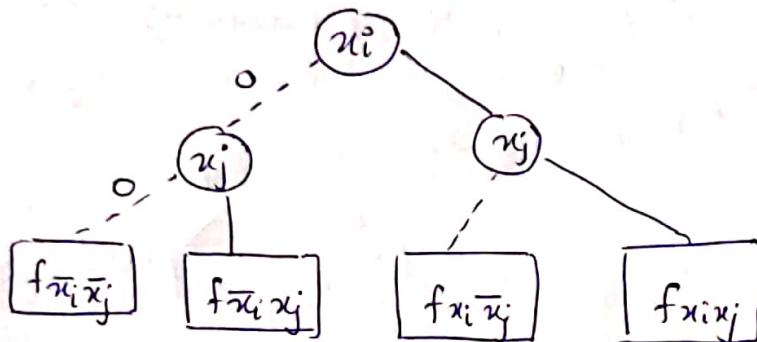
$$f_a = b + c$$

$$f_{\bar{a}} = c$$

Further,

$$f(x_1, \dots, x_n) = x_i x_j f_{x_i x_j} + \bar{x}_i x_j f_{\bar{x}_i x_j} + x_i \bar{x}_j f_{x_i \bar{x}_j} \\ + \bar{x}_i \bar{x}_j f_{\bar{x}_i \bar{x}_j}$$

graphical Rep:



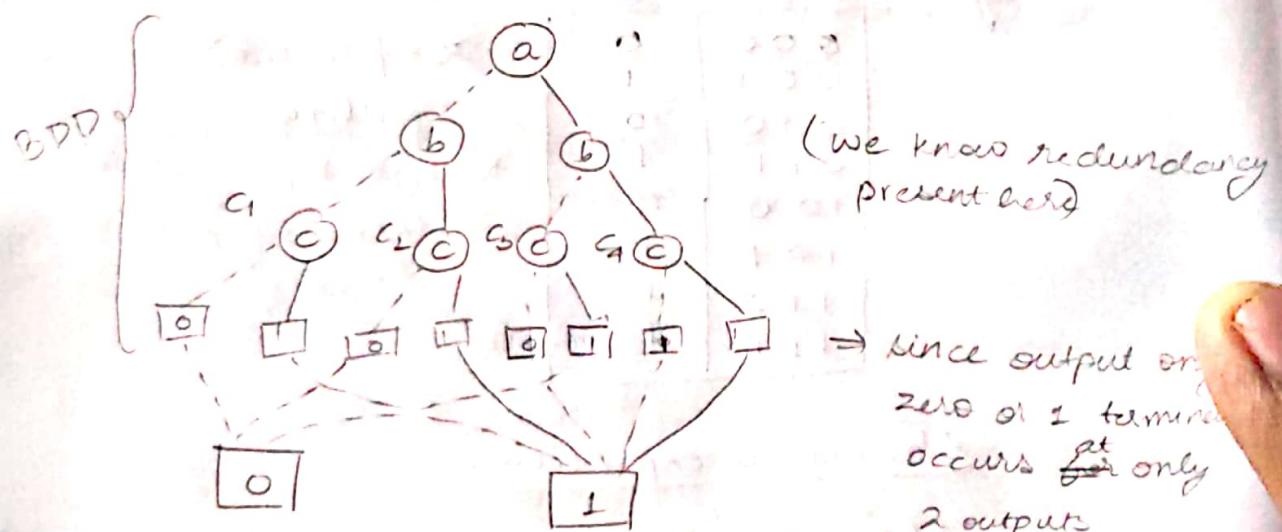
Now, $x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i} = x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i}$

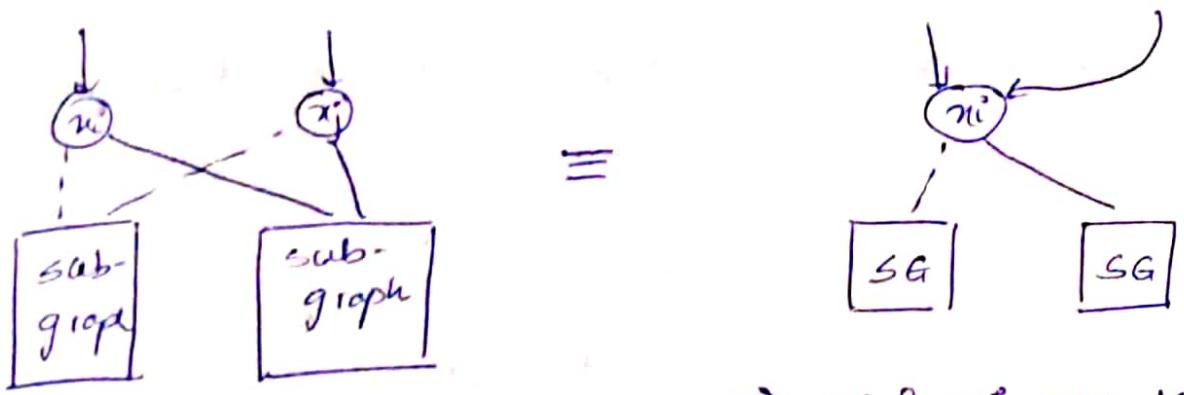
since \downarrow & \uparrow are always complements

so we can build a 'Binary Decision Tree'

$$\Rightarrow 2^{n+1} - 1 \text{ space}$$

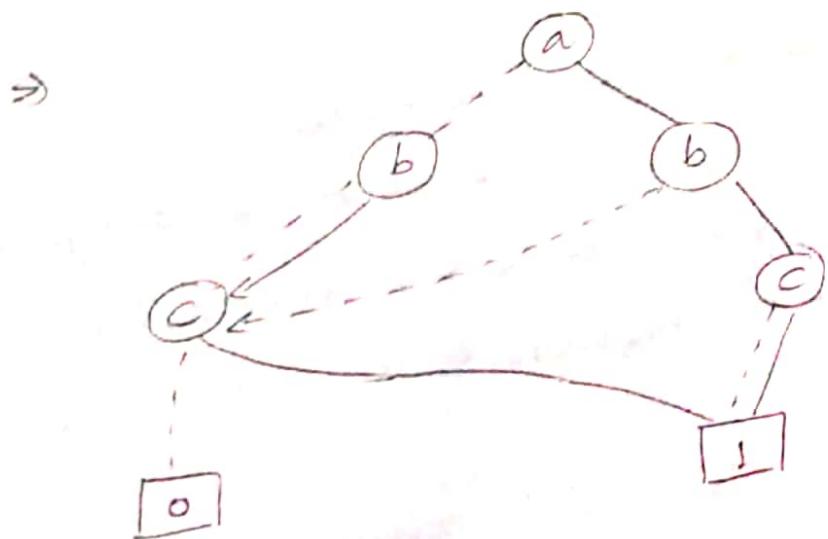
sum of geometric progression





$\Rightarrow u_i \text{ & } x_j \text{ are ISOMORPHIC NODES}$

$\Rightarrow G_1 \text{ & } G_2 \text{ are isomorphic}$



Redundant node \equiv (u_i)



• If $a > b > c \Rightarrow$ Ordered Binary Decision Tree
(which we reduced to a diagram)

⇒ when we use the 3 reductions:

Reduced Ordered ^{Binary} Decision Diagram

(unique canonical form) if order is fundamental.

- 2 functions can't have same ROBDD although OBDD can be same
- if order changes ROBDD changes
- need to choose order which is most compact
- to check equivalence of 2 functions f_1 & f_2 we compare their ROBDD (truth table can also help but not compact)
- since $\boxed{0} = f^{OFF}$, $\boxed{1} = f^{ON}$, thus it contains complete info of truth table
- in ON or OFF any variable can occur once or not at all. Also they occur in the same order

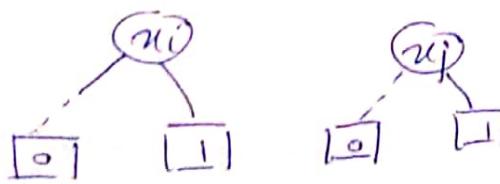
→ For the Reduced diagram

$$f^{ON} = ab + \bar{a}\bar{c}$$

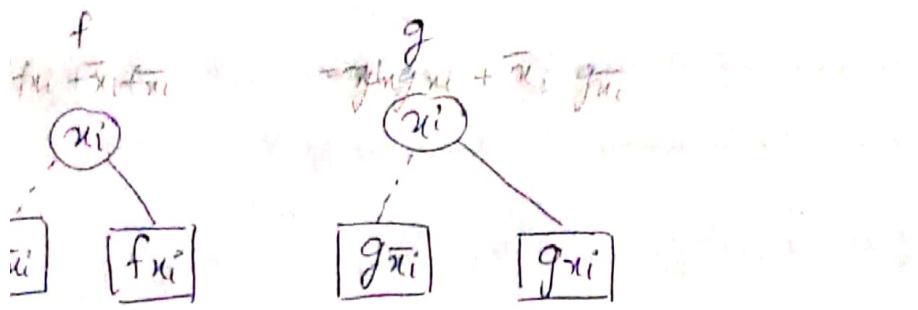
$$f^{OFF} = ab + \bar{a}c$$

→ BDD requires double space than truth table

→ so we need to do an incremental build up

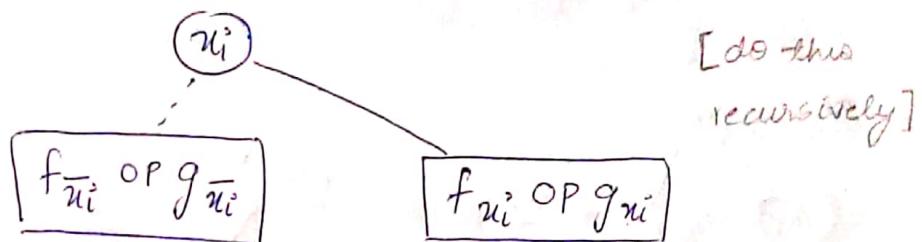


⇒ using these we should be able to generate new

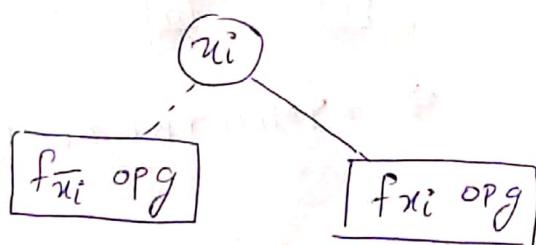


$$f \cdot g = u_i^i f_{u_i^i} g_{u_i^i} + \bar{u}_i^i f_{\bar{u}_i^i} \bar{g}_{\bar{u}_i^i}$$

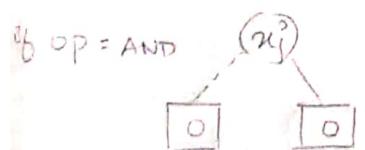
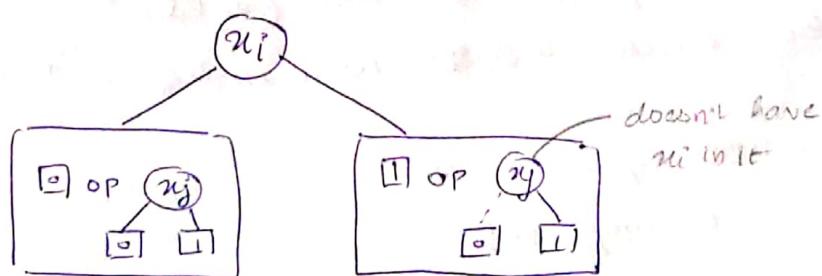
$$f \text{ op } g = u_i^i (f_{u_i^i} \text{ op } g_{u_i^i}) + \bar{u}_i^i (f_{\bar{u}_i^i} \text{ op } g_{\bar{u}_i^i})$$



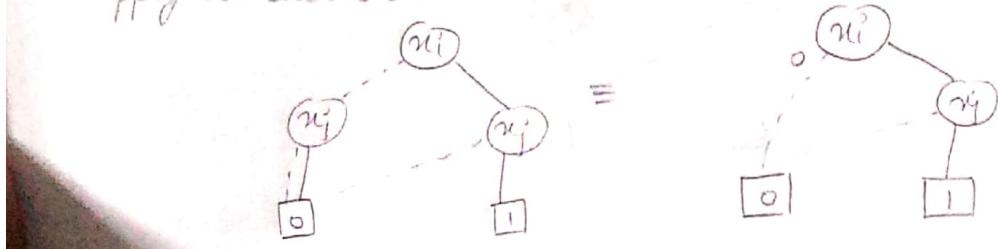
if g doesn't have u_i^i :



\Rightarrow if $u_i^i > u_j$

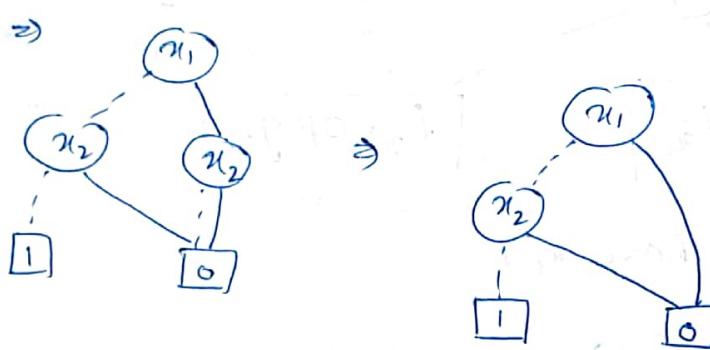
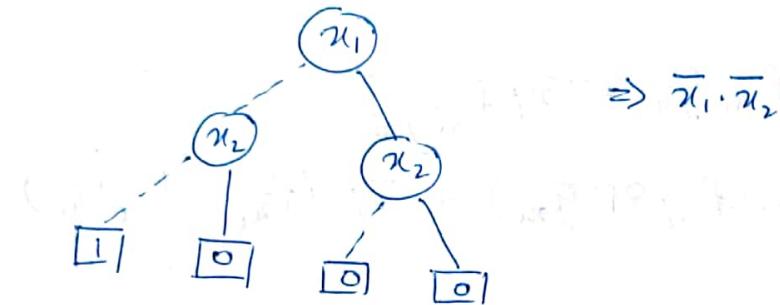


then apply reduction:



- if now we want $\bar{x}_1 \cdot \bar{x}_2$ we just interchange
 - if 1, since outputs now complemented

so for $\bar{x}_1 + \bar{x}_2 = x_1 x_2 + \bar{x}_1 \bar{x}_2 = x_1 \odot x_2$



\Rightarrow SYMMETRICAL FN

$\Rightarrow x_1 \odot x_2 \odot x_3 \odot \dots \odot x_{10}$

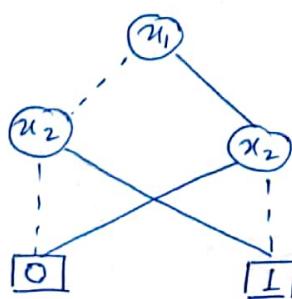
adds only 19 extra nodes for BDD
But truth table has 1024 nodes

each variable adds 2 extra nodes

01/10

- BDD represent ckt
 - i) compactly

eg: $x_1 \oplus x_2$

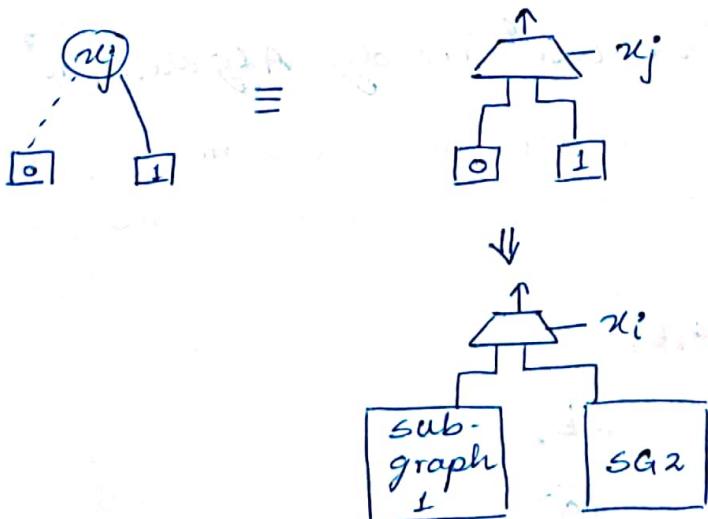


BDD is after every operation apply reduction

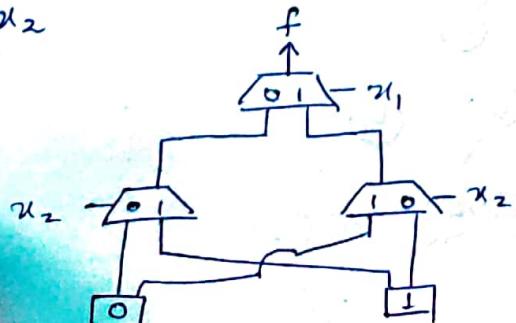
- grows exponentially for multiplier kind of ckt

- $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \dots$ linear growth
- whereas truth table grows exponentially
- ^{BDD}
→ helps to manage the complexity
- BDD helps to compare 2 functions
(can be used to check the correctness)
 - e.g. carry-look ahead adder & ripple carry adder should give the same BDD)
 - Ripple carry adder easier to verify, so it acts as a template to check correctness of other adders
- fully autonomous, human intervention not needed
- compare BDD generated from Boolean & gate level netlist

- Synthesis: directly generate ckt from BDD



$$\rightarrow x_1 \oplus x_2$$



MULTILEVEL
LOGIC

- Size of ckt depends on
 - no. of nodes ($=$ no. of MUX's)
- Performance
 - Max
 - Delay of longest path = $t_m \times \frac{\text{MUX delay}}{\text{no. of variable}}$
- \Rightarrow we want to
 - minimize no. of nodes (area reduction)
 - minimize depth of ckt. (delay reduction)
- No. of nodes depends on order of variables.

Find optimal order of variable that can minimize no. of nodes.

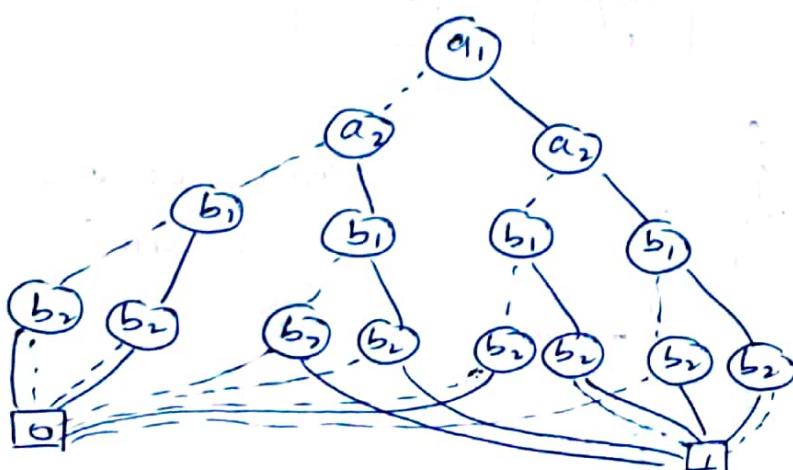
Exploring all order = $n!$

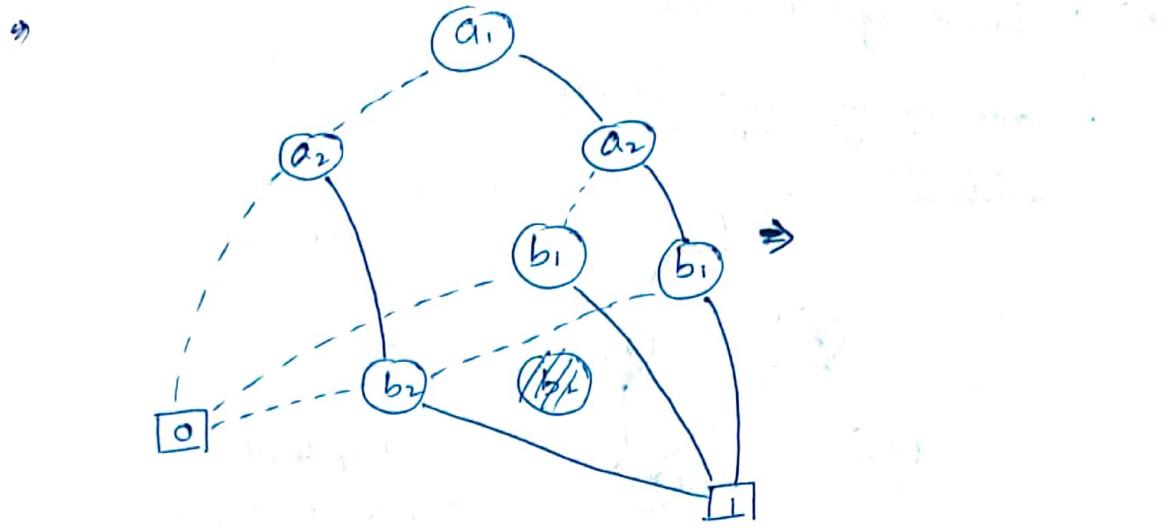
\rightarrow Hence we use 'Greedy Algorithm'

[If not optimum, extra MUX's use more area and power \rightarrow becomes a recurring cost for]

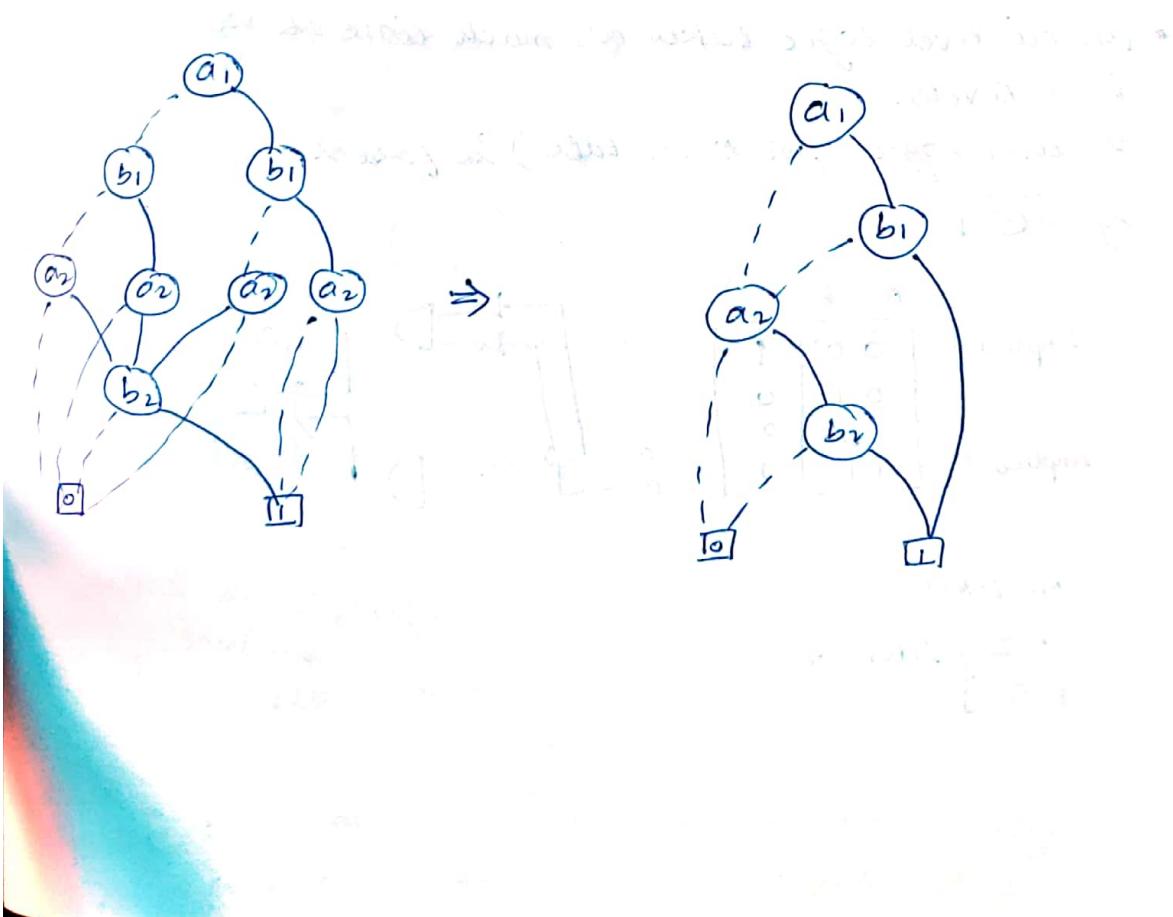
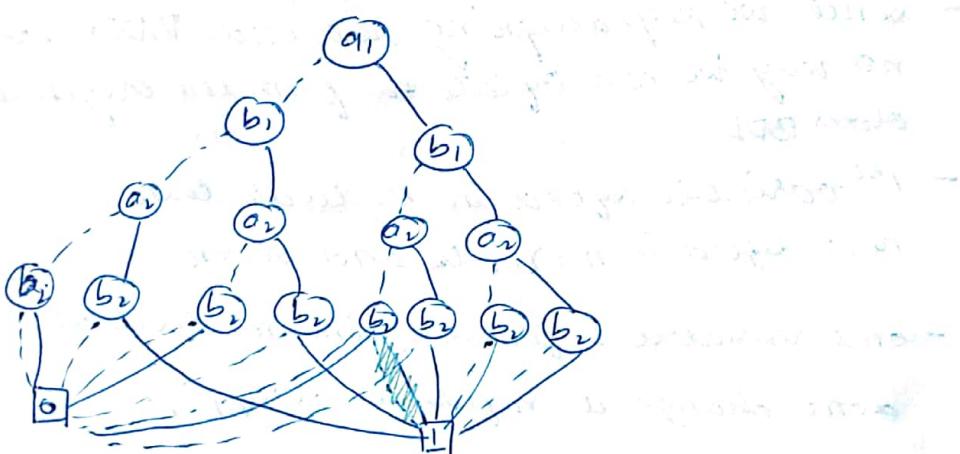
e.g.: $f = a_1 b_1 + a_2 b_2$

a) $a_1 > a_2 > b_1 > b_2$



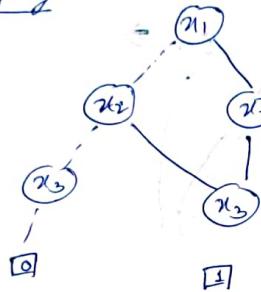


b) $a_1 > b_1 > a_2 > b_2$



To choose optimal order

- swapping algorithm
- sifting



- check using a starting order & check BDD
- then swap x_1, x_2, x_3 if it reduces BDD more than keep that
- Level 4 is fixed, so we keep sifting from level 1 to level 3

- since we keep accepting reduced BDD, so no way we can inflate the from the original order BDD
- 1st variable sifted in n levels, then next sifted in n-1 levels and so on
- once variable 1 finds optimum level we don't change it in further swapping

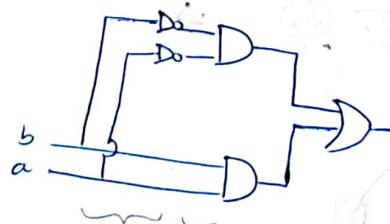
• Multi-level logic takes as much time as we have levels.

2-level logic (for truth table) is faster
eg: $a \overline{\oplus} b$

Implicant	a	b	
Implicant	(0, 0)	1	
Implicant	0, 1	0	
Implicant	1, 0	0	
Implicant	(1, 1)	1	

Variable

a, \bar{a} } literals
 b, \bar{b} } literals



3 levels if all literals (complements) not present

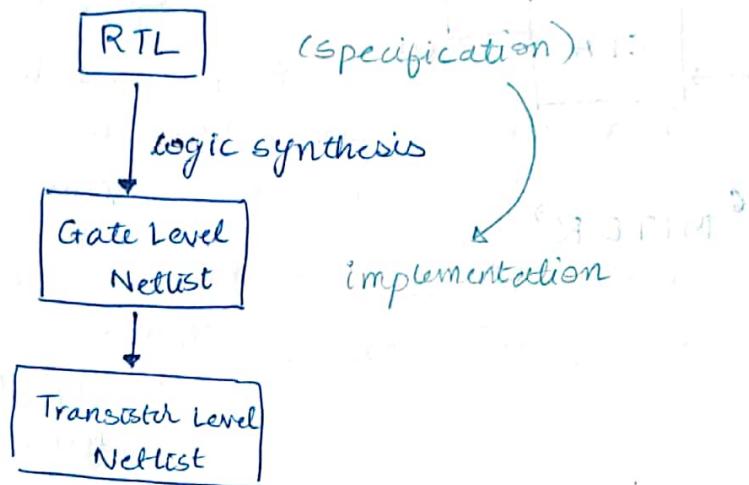
03/10

BDD's :

- store the circuit description in compact format

(check $f_1 = f_2$: part of verification of design)

(as we go from one level of abstraction to another)



- all these levels are mathematically proven to be correct

- Logical Expression

↓
RUBDD

↓ Replace Node by
MUX

↓ Gate level netlist

Do we need to
check for
equivalence?
(RUBDD mathematically
correct)

- Truth Table

↓
minterm

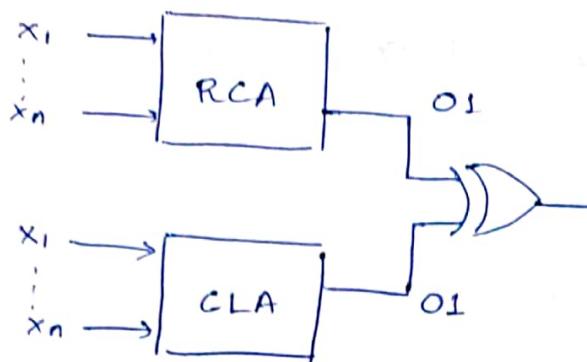
↓ K-Map

↓ CKT

Should be equal since K-Map
reduction is mathematically
correct

- if intermediate steps are mathematically correct then what is the need for testing

- Error in software harder to detect since
software space is large



'MITER'

want it to be 0
for equivalence
if it is 1, then we
know they are

- not equivalent
- tell us the input
for which they aren't
equal
(this input called
COUNTER EXAMPLE)

- Satisfiability problem transforms to

Now Boolean expression can be written as

POS or SOP

e.g. $\underbrace{(a+b+c)}_{c_1} \underbrace{(\bar{a}+\bar{b}+e)}_{c_2} \underbrace{(\bar{a}+\bar{c})}_{c_3} = 1$

check if satisfiable

- computationally harder
to detect

to check equal to 1 POS is better
cos all c_1, c_2, c_3 have to be = 1

→ if we find atleast one set of
inputs such that it equals to
1 → it is satisfiable

→ Equivalent to making a truth table
Need to find another way

→ Assign an order of variable and keep
assigning values in that order

Suppose a, b, c in order

let $a = 0$ then $c_2 = 1, c_3 = 1, c_1 = 0$

$b = 0$ then $c_2 = 1$

$c = 0$ then $c_1 = 0$

⇒ not satisfied

⇒ UNSAT

$c = 1$ then $c_1 = 0$

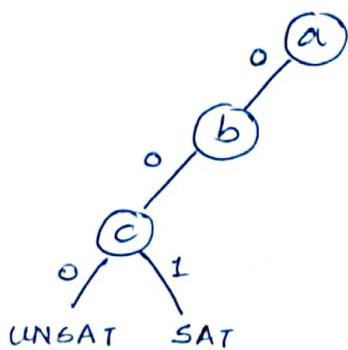
⇒ satisfied

⇒ one of solution is

$$(a, b, c) = (0, 0, 1)$$

→ naive way of solving

⇒ DPLL algorithm



⇒ SAT problem

⇒ c_1, c_2, c_3 ; SAT solver

→ Need to express finite POS

→ To check equivalence of 2 ckt (RCA & CLA).
we want the result to be UNSAT, however,
we need to traverse all possibilities to check
if all are UNSAT

- express the ckt as SAT clause

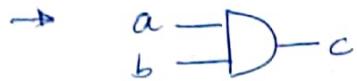
→ replace each gate by their input/output relation and all of them



Behavior: $a \rightarrow b$

$$\bar{a} \rightarrow b$$

$$\Rightarrow (a \rightarrow b) (\bar{a} \rightarrow b)$$



Behaviour: $((a=0) \rightarrow (c=0)) \quad (\bar{a} \rightarrow \bar{c})$
 $((b=0) \rightarrow (c=0)) \quad (\bar{b} \rightarrow \bar{c})$
 $((a=1) \cdot (b=1) \rightarrow (c=1)) \quad (a \cdot b \rightarrow c)$

{ Implication Relationship }
 $x \rightarrow y \equiv \bar{x} + y$

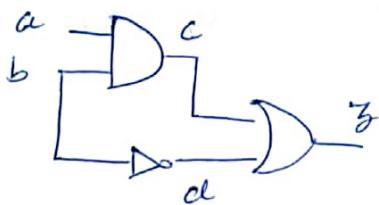
$$\Rightarrow \text{AND} \equiv (a+\bar{c})(b+\bar{c})(\bar{a}\bar{b}+c)$$

$$\text{INVERTER} \equiv (\bar{a}+\bar{b})(a+b)$$

Inverter = 2 clauses

$$\text{AND} = 3 \quad \begin{smallmatrix} - \\ - \\ - \end{smallmatrix}$$

$$\text{XOR} = 4 \quad \begin{smallmatrix} - \\ - \\ - \\ - \end{smallmatrix}$$



$((a=1) \rightarrow (c=1)) \quad (a \rightarrow c)$
 $((b=1) \rightarrow (c=1)) \quad (b \rightarrow c)$
 $((a=0) \cdot (b=0) \rightarrow (c=0)) \quad (\bar{a}\bar{b} \rightarrow \bar{c})$

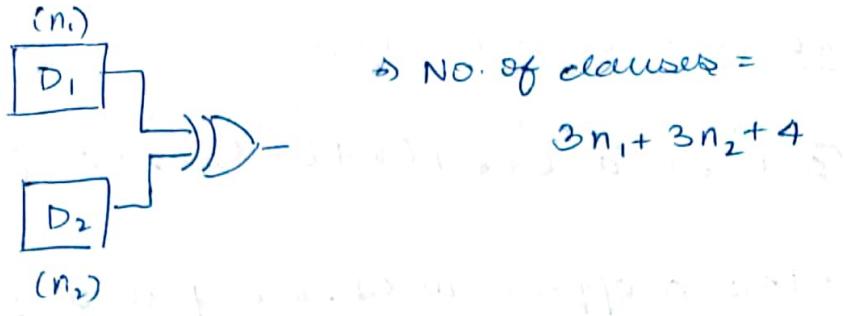
$$\Rightarrow (\bar{a}+c)(\bar{b}+c)(a+b+\bar{c})$$

\Rightarrow Therefore for above ckt,

$$(a+\bar{c})(b+\bar{c})(\bar{a}\bar{b}+c)(\bar{b}+\bar{d})(b+d).$$

$$(\bar{c}+z)(\bar{a}+z)(c+d+\bar{z}) \cdot z$$

\Rightarrow if we have n no. of gates then if we want output = 1
roughly we get $3n$ no. of clauses



1/10 SAT

$$(c_1)(c_2)(c_3) \equiv 1$$

Literal : variable in complemented or uncomplemented form

- DPLL (1960)

↓
assign a value to a variable in some order (alphabetic order)

↓
if fails then backtrack

↳ one of the clause becomes UNSAT

→ naive algorithm

(if UNSAT need to backtrack a lot)

→ Need to accelerate

- ① • Pure literal rule

$$\text{eg: } (a+b+c)(a+\bar{b}+c)(a+\bar{c})$$

we see that 'c' is always uncomplemented, so we assign $a=1$ (or $a=0$ doesn't help to achieve $f=1$)

$$(a+\bar{b}+c)(a+\bar{b}+\bar{c})(a+\bar{c})$$

, here b is also in pure form

$$\Rightarrow a=1, b=0$$

② unit clause rule (1995)

$$(\bar{a} + b + c) \quad (\bar{a} + \bar{b} + e) \quad (b + \bar{c} + d) \quad (\bar{a} + e) \quad (\bar{e} + d)$$

Since a appears in c_1 , we put $a = 1$

$\Rightarrow c_4$ turns into clause with one literal
unit clause

$\Rightarrow e = 1$

$\Rightarrow c_5$ turns into a unit clause

$\Rightarrow d = 1$

\rightarrow propagate these unit clauses, called
'Boolean constraint Propagation'
(BCP)

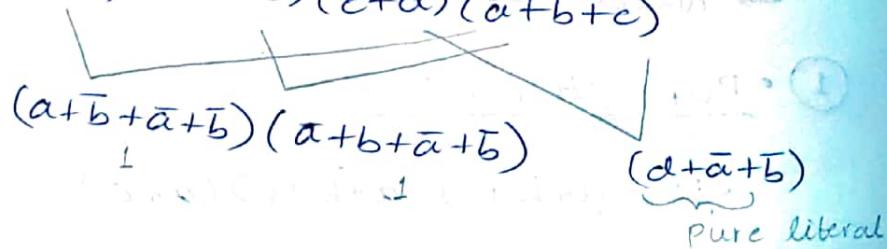
\rightarrow so rather than assigning in
chronological order, we use BCP

③ consensus theorem

$$(a+b)(\bar{a}+c) = (a+b)(\bar{a}+c)(b+c)$$

\rightarrow either one of b or c should be 1

eg: $(a+\bar{b}+\bar{c})(\bar{a}+b+\bar{c})(\bar{c}+d)(\bar{a}+\bar{b}+c)$



\rightarrow if either end up in pure literal
clause or use consensus again

④ stallman's Method [satisfying literal
by literal]

eg: $(a+b)(\bar{a}+c)(\bar{b}+d)(\bar{c}+d) = 1$

Satisfying:

$c_1 \Rightarrow a=1$ or $b=1$ (literal based algorithm)

{ If there are multiple possibilities then we explore the tree, else no point in wasting time }

If we choose $a=1$

$c_2 \Rightarrow c=1$

$c_4 \Rightarrow d=1$

$\Rightarrow SAT$

if $a=0$

$c_1 \Rightarrow b=1$

$c_2 \Rightarrow satisfied$

$c_3 \Rightarrow d=1$

$c_4 \Rightarrow satisfied$

$\Rightarrow SAT$

→ we have now 2 set of values

intersection: $d=1$

$\Rightarrow d=1 = \underline{\text{Essential Assignment}}$

{ Pick one variable, and find the value set for it being equal to 1 and 0, then take intersection }

⑤ Recursive learning (RL)

- does similar job but clause by clause

e.g.: $(a+b)(\bar{a}+c)(b+d)(\bar{c}+d) \equiv 1$

for satisfying C1

$$\rightarrow a=1 \Rightarrow c=1 \Rightarrow d=1$$

$$\rightarrow b=1 \Rightarrow d=1$$

$$y_7 = d=1$$

for satisfying C2

$$\rightarrow a=0 \Rightarrow b=1 \Rightarrow d=1$$

$$\rightarrow c=1 \Rightarrow d=0$$

$$y_7 = d=1$$

essential
assign

⇒ Build tree for all variables that are not essential assignment

⑥ Local Search Method

- Suppose we do a random assign

$$a=1, b=1, c=1, d=0$$

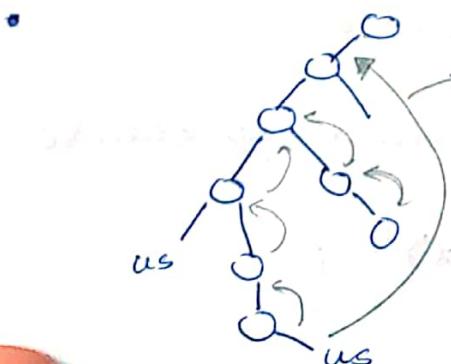
⇒ C4 is UNSAT

→ C3 also

⇒ Need to flip some variable

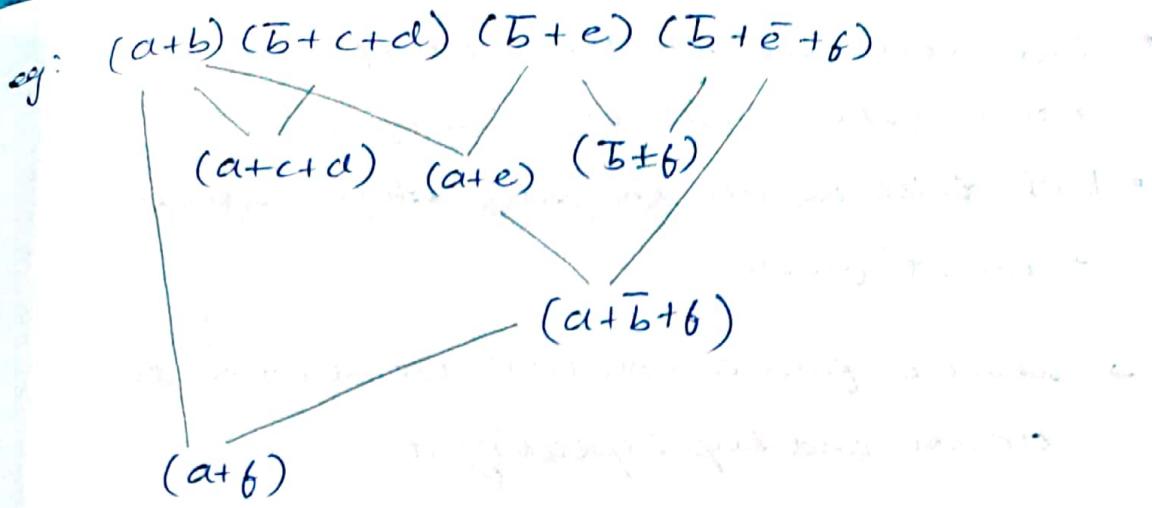
⇒ we flip that variable which appears in more no. of clauses

- Find essential ^{variable} assignments and assign them high priority then explore the tree
- Simple approach would be to find the variable which appears in most clauses and prioritize it

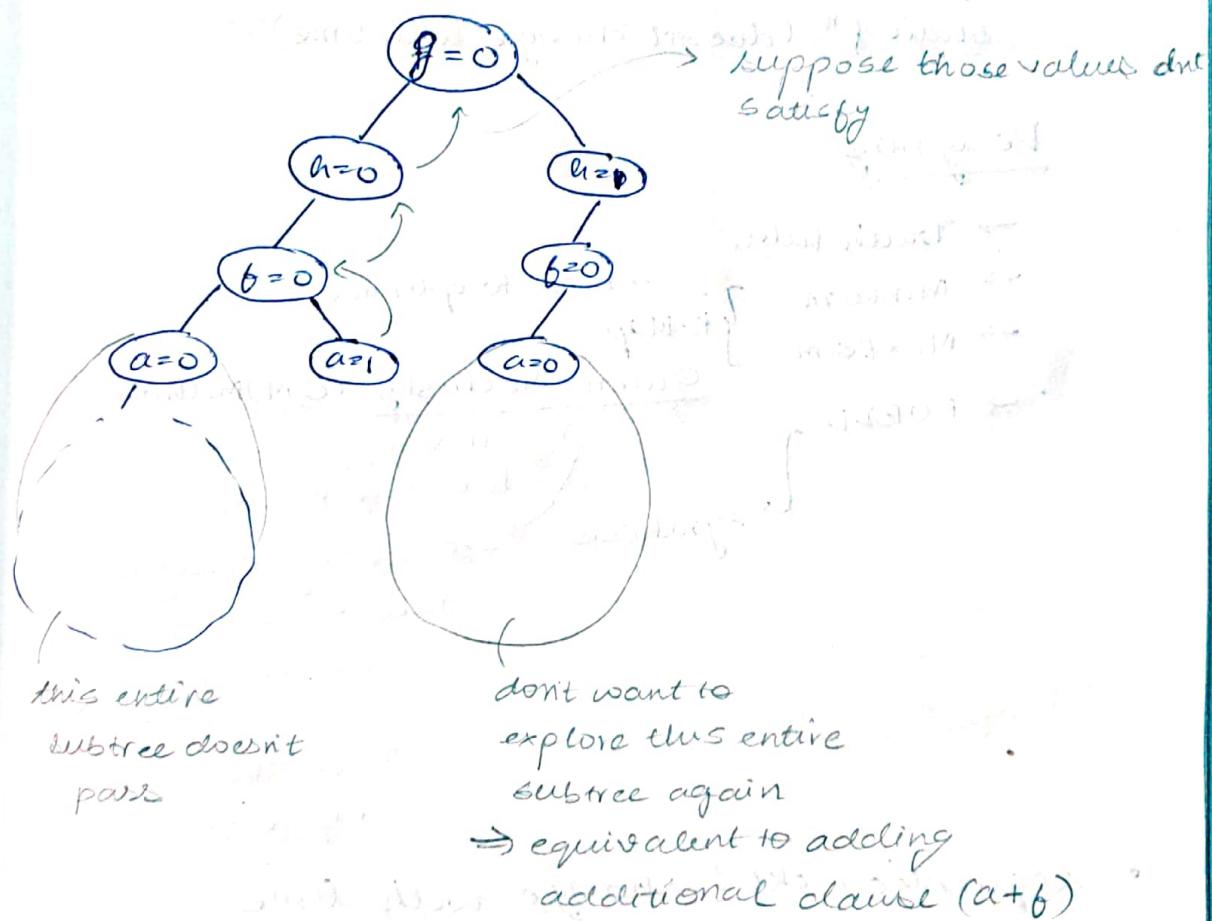


Some way to get to the root problem rather than going chronologically

Non-chronological
Backtracking'



→ consensus helps us make additional clause



⇒ 'clause recording'

↳ once recorded we don't make a wrong assignment

- when expansion is linear we use BDDs
- Sat Solvers: Zchaff
Minisat

- SAT tells us whether satisfiable or not i.e. one counterexample
- BDD tells us all the sets when satisfiable + unsatisfiable
- when we find a solution, we record that clause, and keep repeating it

8/10 SAT Solver

$$f_1 \equiv f_2$$

static f^n (doesn't change with time)

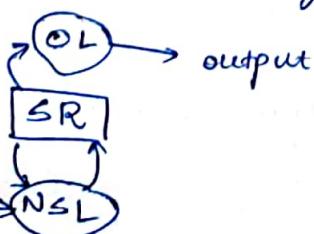
Designing

- Truth table
 - Minterm
 - Maxterm
 - RORDD
- } need to optimize,
K-Map
- Quinn-McClusky (QM) Method
- } synthesis
- (review yourself)
Tabular method
Set covering problems
(need to select)
 - essential prime implicant (EPI)
 - selective prime implicant

- Sequential ckt: change with time

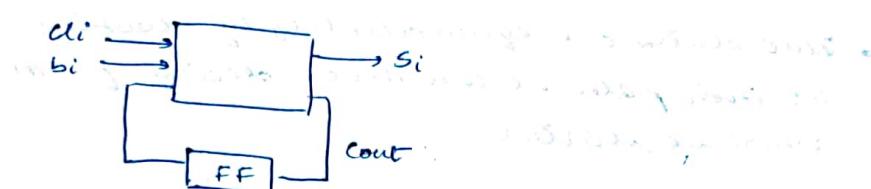
- need FSM

- state
- output logic
- next stage logic



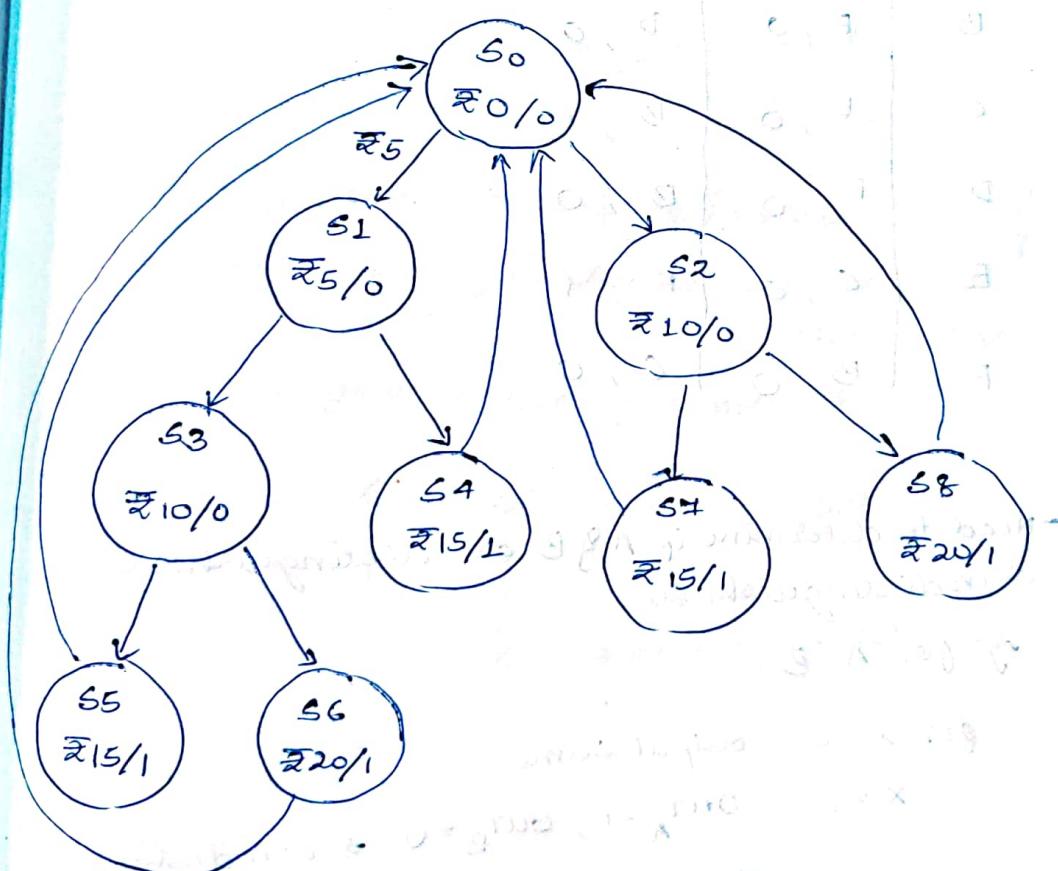
• Adder:

- can replicate a full adder in series (like RCA)
- or make a sequential adder



⇒ output depends on history

e.g: we have a coffee dispenser machine



Hence finally we want to dispense coffee,
states S_5, S_6, S_4, S_7, S_8 are the same
hence we can minimize them

⇒ state Minimization Problem

→ Need to build the notion of state first
for a given problem

→ State minimizatn should occur automatically

- Two states are equivalent if by looking at their path we can never decide from where we started

States	Nextstate, output (z)	
	$x=0$	$x=1$
A	E, 0	D, 1
B	F, 0	D, 0
C	E, 0	B, 1
D	F, 0	B, 0
E	C, 0	F, 1
F	B, 0	C, 0

- Need to determine if A & B are distinguishable or indistinguishable

e.g. for A, B

for $x=0$ output same

$x=1$ $OUT_A = 1, OUT_B = 0 \Rightarrow$ can distinguish

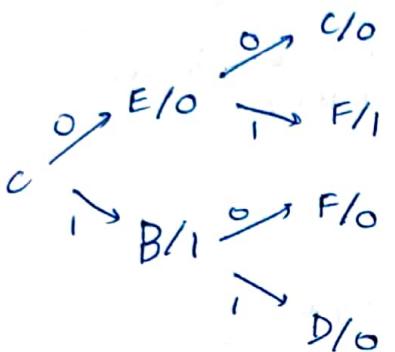
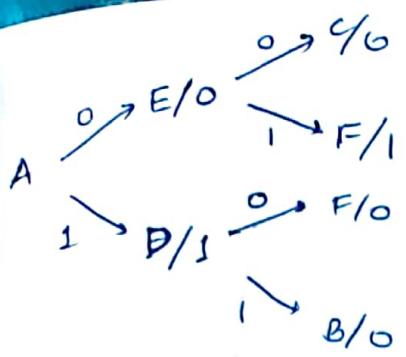
⇒ 1 distinguishable

A & C ⇒ 0 equivalent

$x=0$ same if not 1 distin.

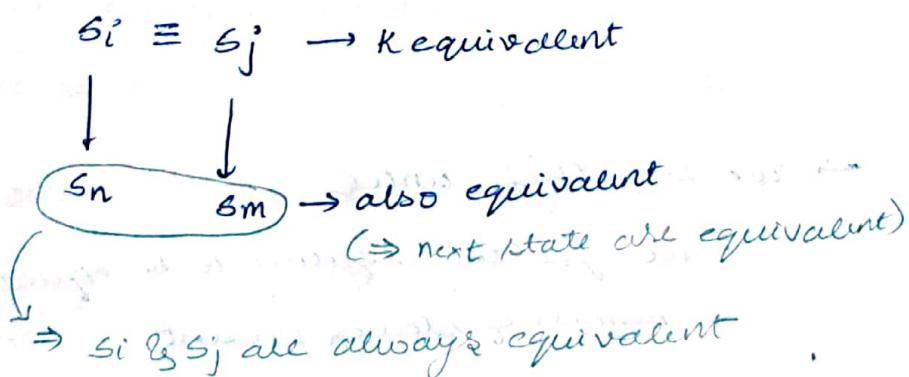
$x=1$ same

⇒ 1 indistin



2 equivalent

- K distinguishable $\Rightarrow K-1$ equivalent
- g_0 never indistinguishable \Rightarrow always equivalent
 \rightarrow But go on indefinitely to notice them



- In the beginning all the states (w/o applying any input) are equivalent

• Partitioning

$$P_0 = (A B C D E F)$$

$$P_1 = (A \times E) / (B \times E)$$

$$P_1 = (ACE)(BDF)$$

KIDBF DBC
indistinguishable

by applying 1

input

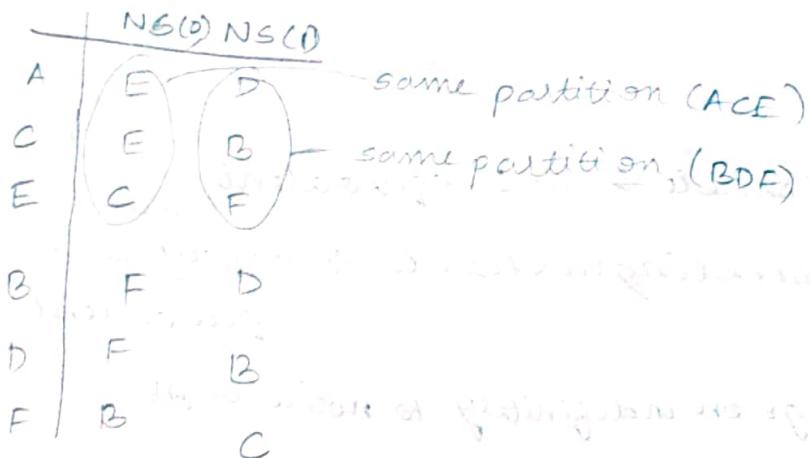
ACE) (BD) (F) *whose next state lies in the same partition*

$$P_3 = (AC)(E)(BD)(F)$$

$$P_A = (A \cap E)$$

$$P_4 = (AC)(E)(BD)(F) \quad \text{↳ keeps repeating loop here}$$

Taking the example of P_2



→ we run this until

- we get the sequence to be equal
 - number of states present

- checking equivalence of 2 state machine

e.g.: 2 machines

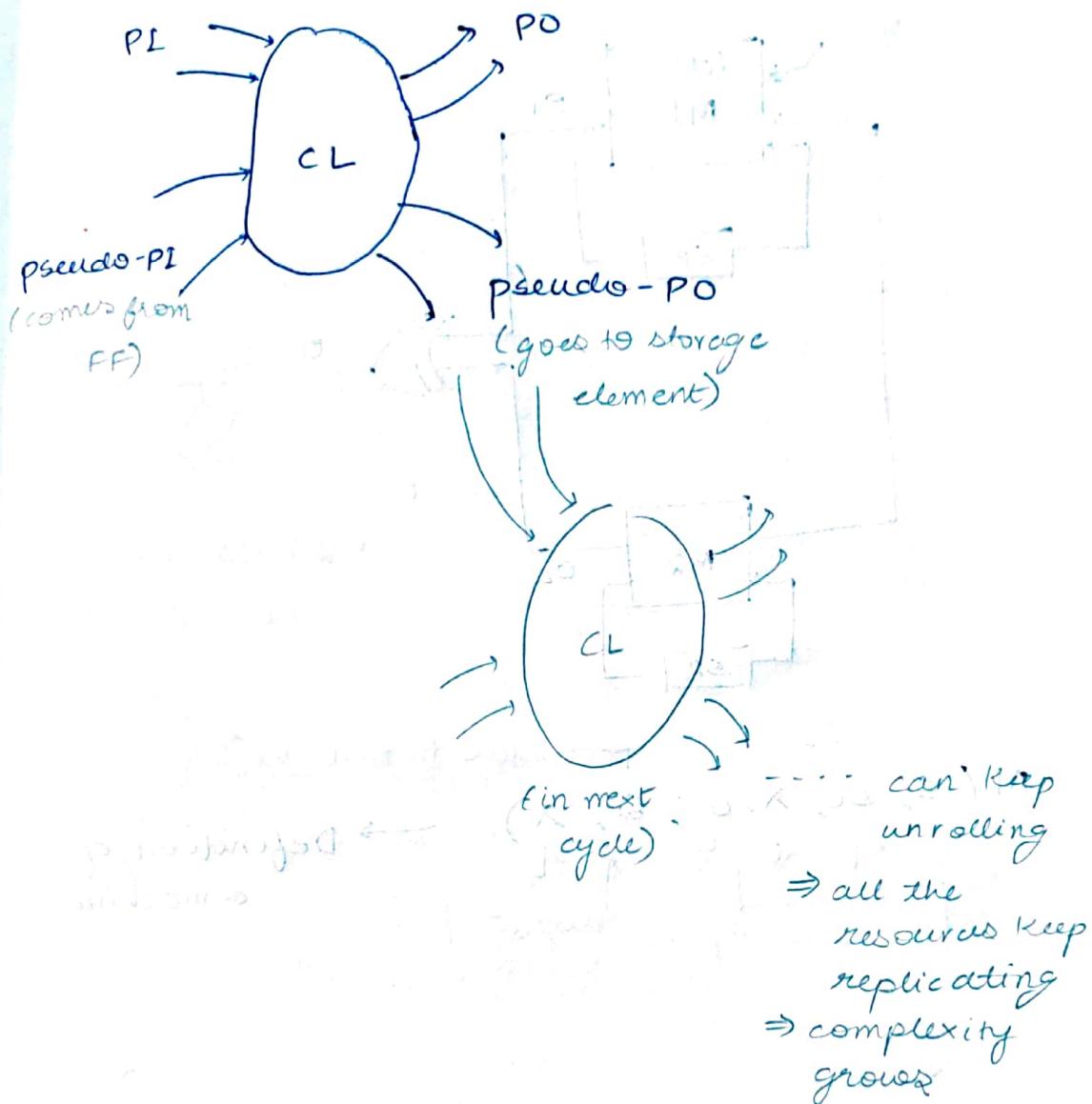
M
↓
 M'_i

M₂

↓ minimize machine

M_2' \Rightarrow Isomorphic (check)

Isomorphic : if 2 graphs are same



we start from a known state

→ reset state

{ Unrolling = Time Frame Expansion }

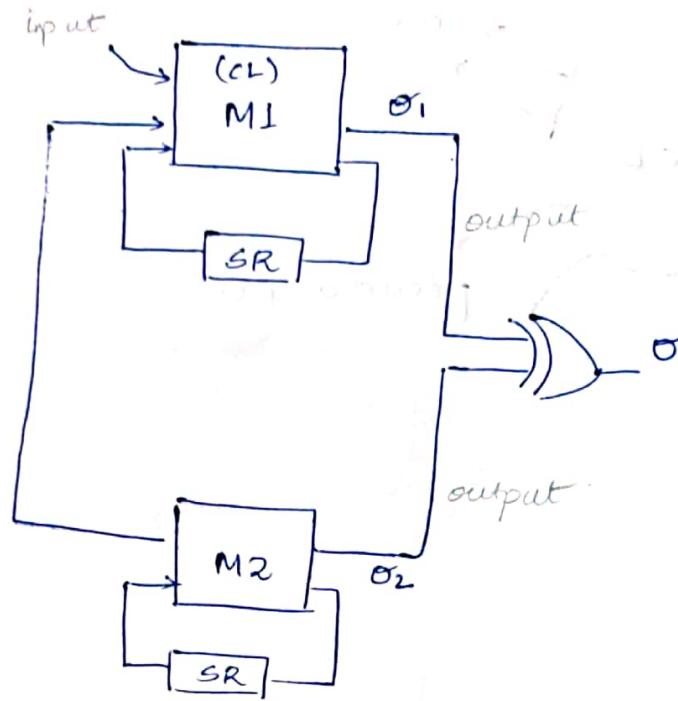
If we have m outputs and we unroll n no. of times

→ $m \times n$ outputs

→ which need to be XOR'd by the Miter

→ we don't roll for a long period of time
we keep a boundary
and check for equivalence using BDD

⇒ 'Sequential Equivalence Checking'



$M(s, s_0, x, o, f^n)$ → Definition of
 state transition f^n
 initial state
 inputs
 output
 variable output f^n
 state
 variable
 (set of states)

$$M_1: (s', s'_0, x, o, s', \lambda)$$

$$M_2: (s^2, s_0^2, x, o, s^2, \lambda^2)$$

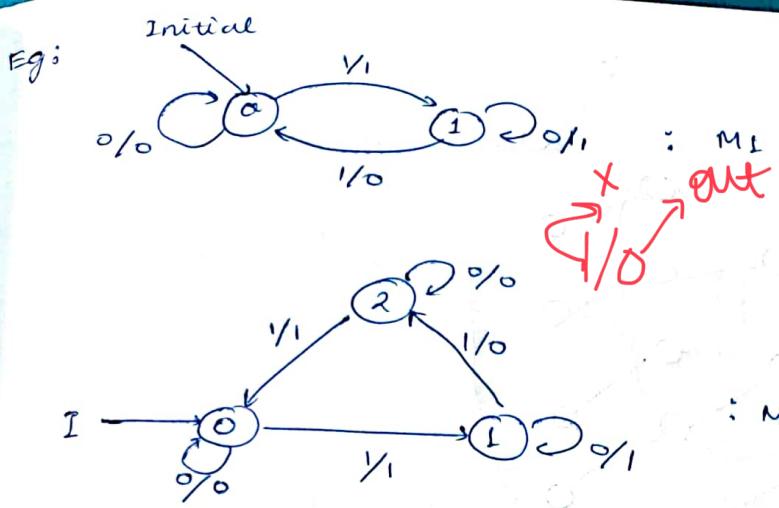
$$\rightarrow \text{Total number of states} = |s'| \times |s^2|$$

$$O = \begin{cases} 0 & \text{when } O_1 = O_2 \\ 1 & \text{o.w.} \end{cases}$$

hence called
PRODUCT
MACHINE.

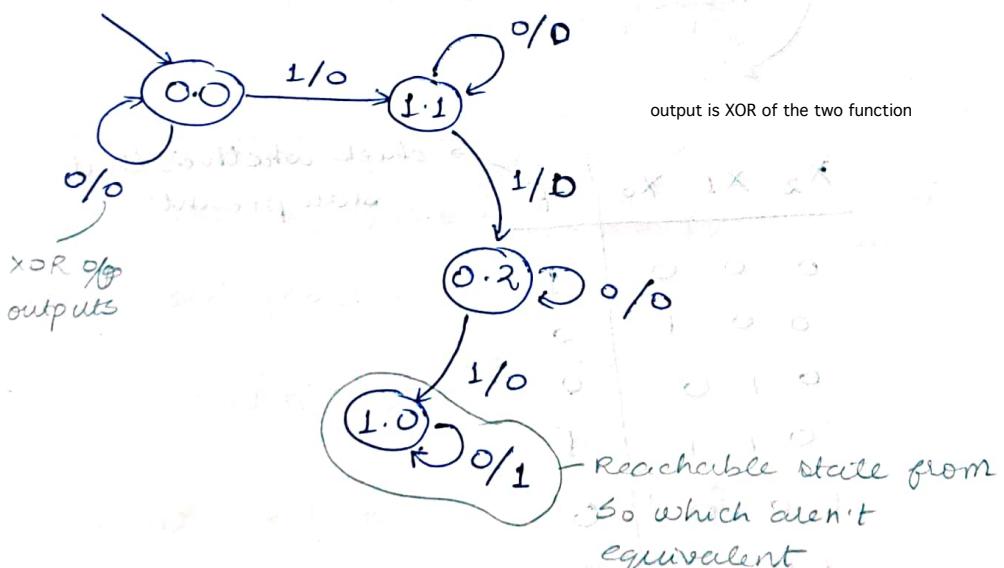
→ we do a reachability analysis

whether all states are
reachable or not



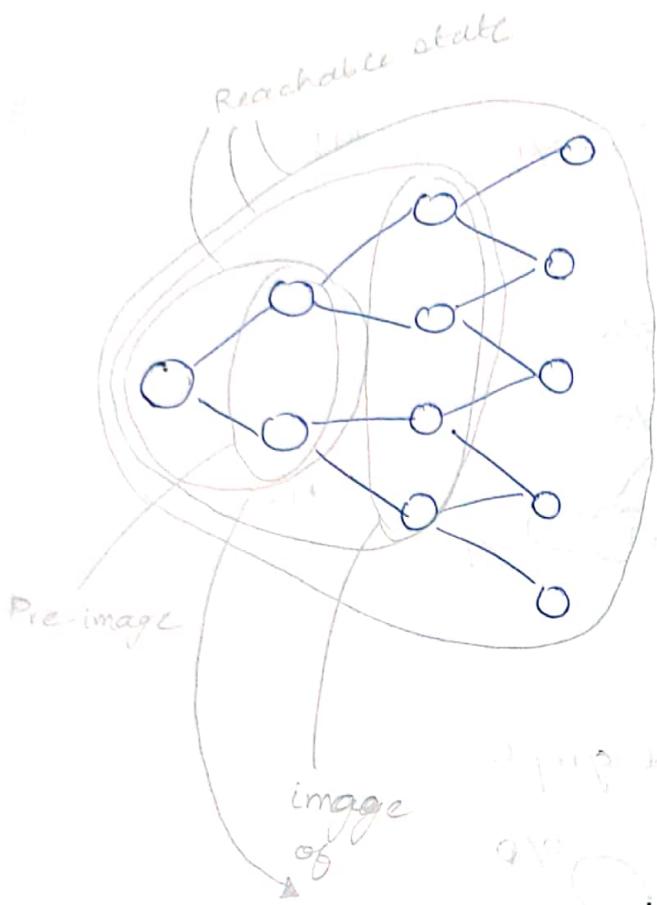
Total states = 6

→ Making the combined graph



⇒ 1110 input helps get here

→ we continue giving input till we keep getting new states.
If we don't find new states (we stop) and output is always 0 then both are equivalent



x_2	x_1	x_0	$f(x)$	check whether that state present
0	0	0	1	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	1

Suppose we get a set of states S and make its ROBDD. Next we add another set state to that state set and OR the 2 ROBDD

x	R_1	δ_0	t_1	t_0	F	→ checking which transition is valid
0	0	0	0	0	1	
0	0	0	0	1	0	
0	0	0	1	0	0	
0	0	0	1	1	0	
1	0	0	0	0	0	
1	0	0	0	1	1	
1	0	0	1	0	0	
1	0	0	1	1	0	

→ set of reachable states

Transition Relationship: $R(x, \delta, t)$

$C(\delta)$: Set of states from where we can start

$$T(x, \delta, t) = R(x, \delta, t) \cdot C(\delta) \rightarrow \text{set of reachable states from } C(\delta)$$

- Eliminating a variable

$$f(x_1, x_2, \dots, x_i, \dots, x_n) = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i}$$

if we eliminate x_i in $f(x_1, \dots, x_n) = (n-1)^n$

$$f = f_{x_i} + f_{\bar{x}_i}$$

eliminate x & then δ and get $T(t)$

getting a unique image

$$\exists_{x_i} f(x_1, \dots, x_n) = f_{x_i} + f_{\bar{x}_i}$$

Existentially quantify

$$T(x, \delta, t) = \exists_x \exists_{\delta} T(t)$$

then to the set of reachable states

- Backward reachability: Backtracking from some state to initial state

- we might get 2 states which are not equivalent. However, if we are not able to reach that state from the initial state then those machines are all equivalent

10/10 Equivalence checking of 2 state machine

$$M_1(S^1, S_0^1, X, O, S^1, \lambda) \equiv M_2(S^2, S_0^2, X, O, S^2, \lambda)$$

Reachability Analysis (in a Product machine)

$$M = M_1 \times M_2 (S^P, S_0^P, X, O, S^P, X^P)$$

$$\Theta = \begin{cases} 0 & \text{if } (\theta_1 = \theta_2) \\ 1 & \text{if } \theta_1 \neq \theta_2 \end{cases}$$

- can store as a boolean function

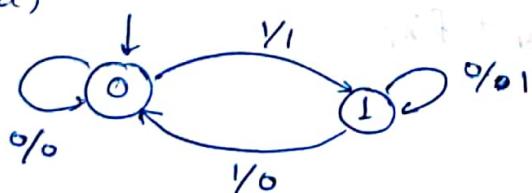
↳ store as ROBDD

$C(\alpha)$ = set of all reachable states by that time

Transition Function:

↳ $R(X, Q, t)$
represent as ROBDD

e.g.; (a)



s_0 = current state = $\{0, 1\}$

t_0 = next state = $\{0, 1\}$

x = input = $\{0, 1\}$

s'_0 = initial state

= $\overline{s_0}$

Current state - from current state to next state

x	0	1
0	0	1
1	1	0

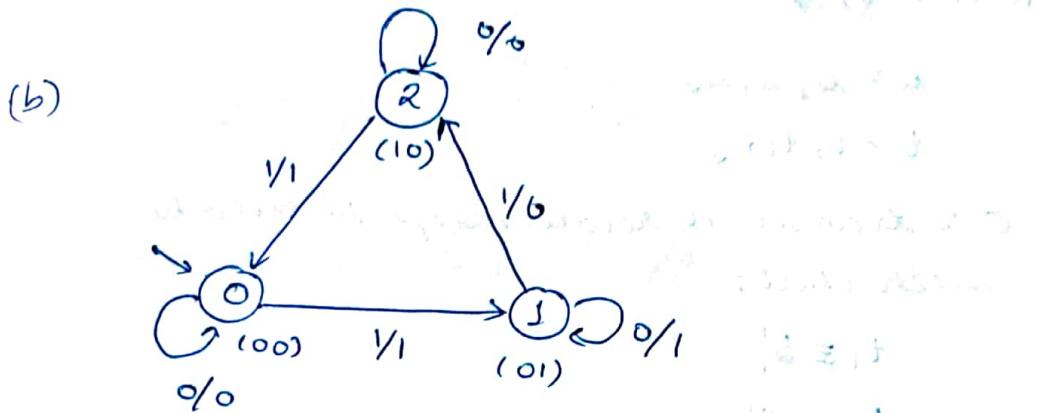
$$\Rightarrow S'(x, \alpha) = \bar{x}\alpha_0 + x\bar{\alpha}_0$$

(next state)

x	0	1
0	0	01
1	1	0

$$\Rightarrow \lambda(x, \alpha) = \bar{x}\alpha_0 + x\bar{\alpha}_0$$

(output)



If we are comparing with above machine,
we need same input, output

$$\Rightarrow x = \alpha_0, z = \alpha_1, \lambda = (\dots, \lambda) \text{ etc}$$

$$S_0^2 = \bar{\alpha}_2 \bar{\alpha}_1$$

writing the LSB of state

$\alpha_2 \alpha_1$	0	1
00	0	1
01	1	0
11	x	x
10	0	0

no fourth state

$$\Rightarrow S_1^2 = \bar{\alpha}_2 \bar{\alpha}_1 x + \alpha_1 x$$

next state MSB

$\alpha_2 \alpha_1$	0	1
00	0	0
01	0	1
11	x	x
10	1	0

$$\Rightarrow S_2^2 = \alpha_2 \bar{x} + \alpha_1 x$$

x_2x_1	0	1
00	0	1
01	1	0
11	x	x
10	0	1

$$g^2 = \alpha_2 \bar{x} + \beta_2 x$$

$R(x, s, t) \Rightarrow$ Relation between two graphs

$$\alpha = \alpha_2 \alpha_1 \alpha_0$$

$$t = t_2 t_1 t_0$$

our transition relationship should be such that:

$$t_1 \equiv s_1'$$

$$t_2 \equiv s_1^2$$

$$t_3 \equiv s_2^2$$

$$\Rightarrow R(x, s, t) = (t_1 \equiv s_1') \cdot (t_2 \equiv s_1^2) \cdot (t_3 \equiv s_2^2) \\ = \prod_i (t_i \equiv s_i)$$

[Equivalence is XNOR operation

$$a \oplus b \Rightarrow a \oplus b = ab + \bar{a}\bar{b}$$

$$= (t_1 \equiv \alpha_0 \bar{x} + \beta_0 x) \cdot (t_2 \equiv \alpha_2 \bar{\alpha}_1 x + \beta_2 \bar{x}) \\ (t_3 \equiv \alpha_1 x + \beta_1 \bar{x})$$

$$C(\alpha) = \bar{\alpha}_2 \bar{\alpha}_1 \bar{\alpha}_0 (000)$$

$$T(x, s, t) \Rightarrow R(x, s, t) \cdot C(\alpha)$$

[We know the BDD of individual parenthesis and hence can combine and reduce]

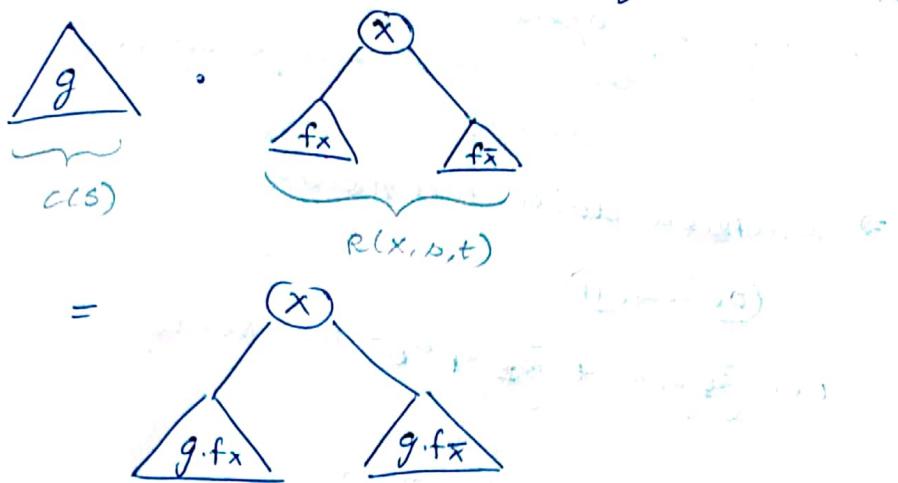
$$T = (\bar{\alpha}_2 \bar{\alpha}_1$$

$$T = (t_1 \cdot$$

$$\text{at } x = 0$$

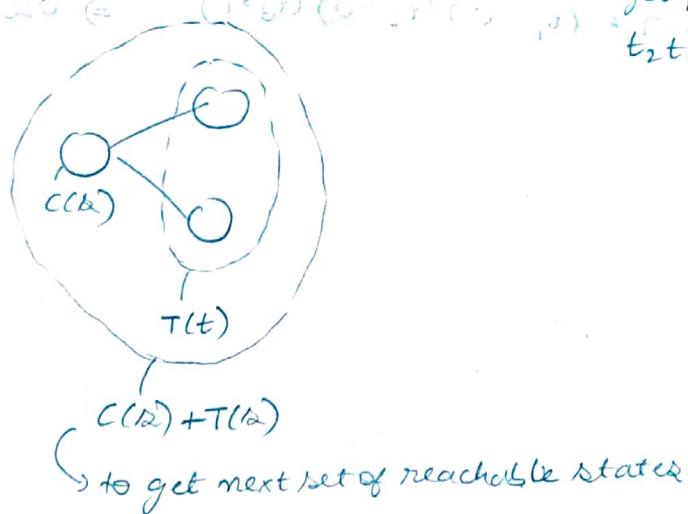
$$x = 1$$

want to existentially quantify x



$$\Rightarrow \exists_x T(x, b, t) = \triangle g \cdot f_x + \triangle g \cdot \bar{f}_x = \triangle g$$

(Further eliminate \$g\$ to get \$f^{\bar{n}}\$ of \$T(t)\$)



$$T = (\bar{\alpha}_2 \bar{\alpha}_1 \bar{\alpha}_0) (t_1 \equiv \alpha_0 \bar{x} + \bar{\alpha}_0 x) (t_2 \equiv \bar{\alpha}_2 \bar{\alpha}_1 x + \alpha_1 \bar{x}) \\ (t_3 \equiv \alpha_1 x + \alpha_2 \bar{x})$$

$$T = (t_1 \equiv \bar{\alpha}_2 \bar{\alpha}_1 \bar{\alpha}_0 x) (t_2 \equiv \bar{\alpha}_2 \bar{\alpha}_1 \bar{\alpha}_0 x) (t_3 \equiv 0)$$

$$\text{at } x=0 : T = (t_1=0)(t_2=0)(t_3=0) \Rightarrow 00$$

because currently we are in $S_2 S_1 S_0$,
which should be 1 if we are in this state

- Above is the formal procedure for combining 2 graphs
(compared to the graph combining we did by hand calculation)

⇒ Reachable states till now =



i.e. $\overline{b_2} \overline{b_1} \overline{b_0} + \overline{b_2} b_1 b_0 = \text{new state}$

generated by
old state generated by
new states

$$\begin{aligned} T &= (\overline{b_2} \overline{b_1} \overline{b_0} + \overline{b_2} b_1 b_0) R(x, s, t) \\ &= (t_1 \equiv \overline{b_2} \overline{b_1} \overline{b_0} \bar{x} + \dots) (t_2 \equiv \overline{b_2} b_1 b_0 \bar{x} + \dots) \\ &\quad (t_3 \equiv \overline{b_2} b_1 \overline{b_0} x) \end{aligned}$$

at $x=0$: $T = (t_1=1)(t_2=1)(t_3=1) \Rightarrow 11$
 $x=1$: $T = (t_1=0)(t_2=0)(t_3=1) \Rightarrow 02$